University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
  - Homework 6: this Friday @ 11:59 pm
  - Lab 5: Tuesday 2/28 @ 11:59 pm
  - Assignment 2: Friday 3/17 @ 11:59 pm
    - Support video and slides on Canvas
- Midterm Exam on Wednesday 3/1
    - in-person, closed-book
    - Study guide and old exams on Canvas
    - Practice questions on GradeScope (with answers)
- Lost points because autograder or simple mistake?
  - please reach out to Grader TA over Piazza
- Navigating the Panopto Videos
  - Video contents
  - Search in captions

Previous lecture

- Huffman Compression

This Lecture

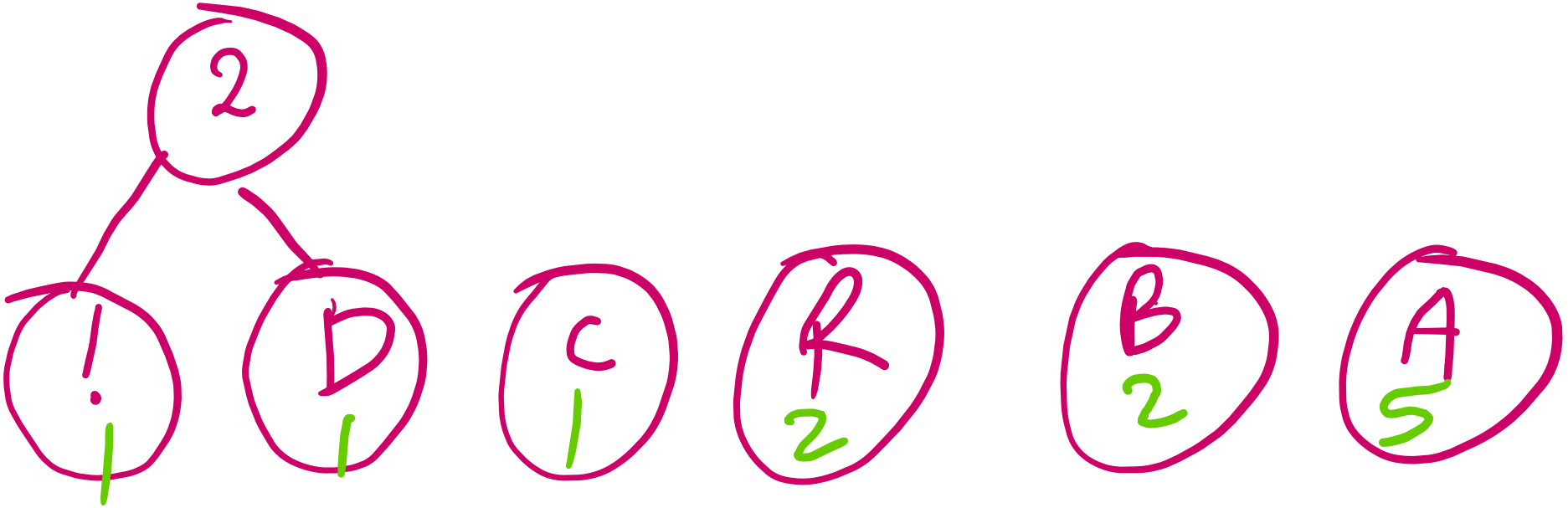
- Huffman Compression
- Run-length Encoding
- LZW

Huffman Compression Example

- Build a tree for “ABRACADABRA!”
- file size: *n* = *12*
- no. of unique characters: *K* = *6*

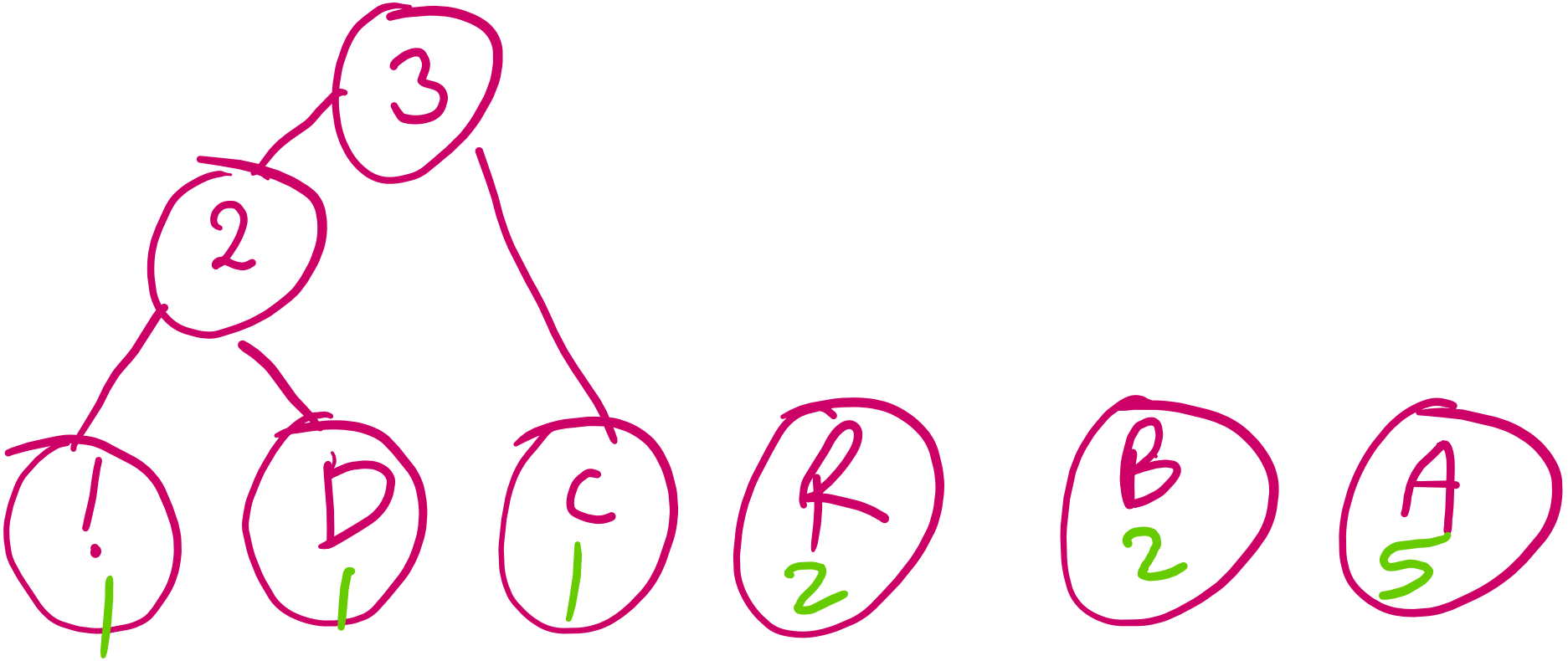
Example

- Build a tree for “ABRACADABRA!”



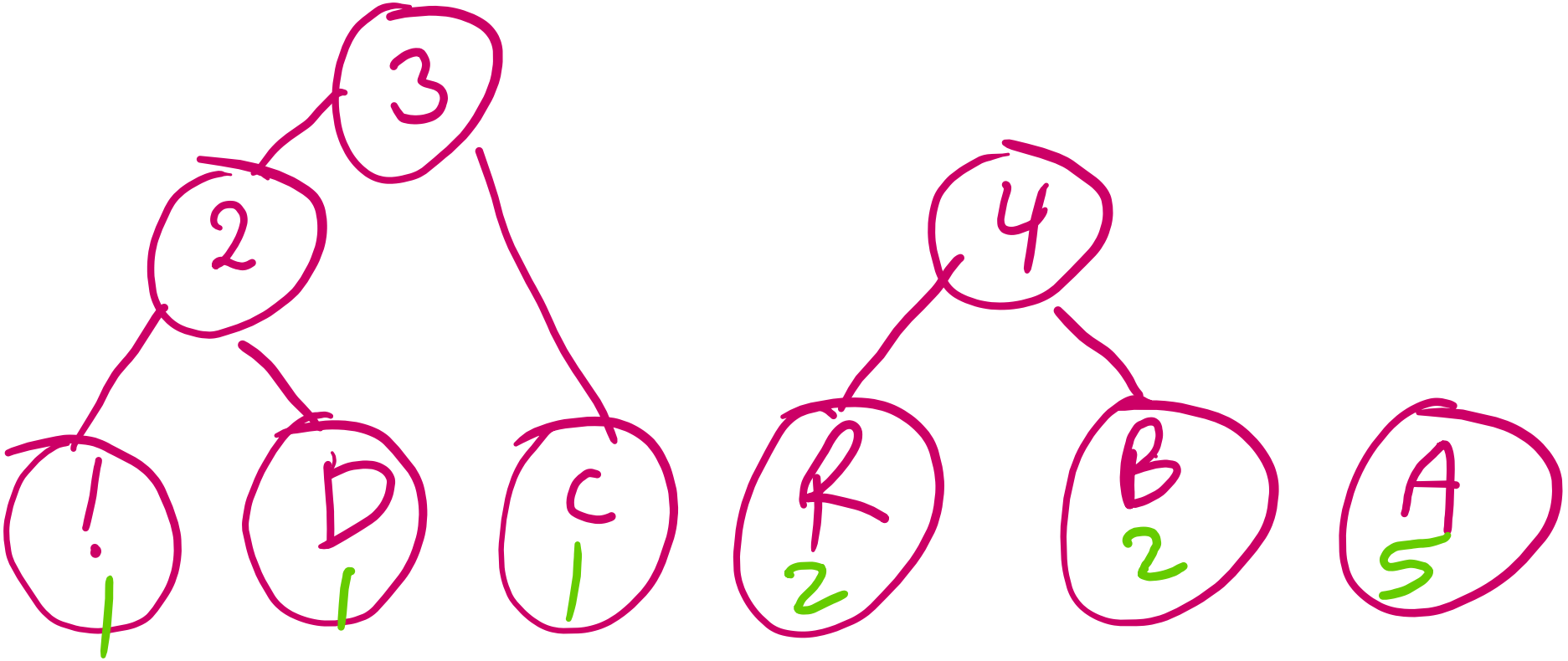
Example

- Build a tree for “ABRACADABRA!”



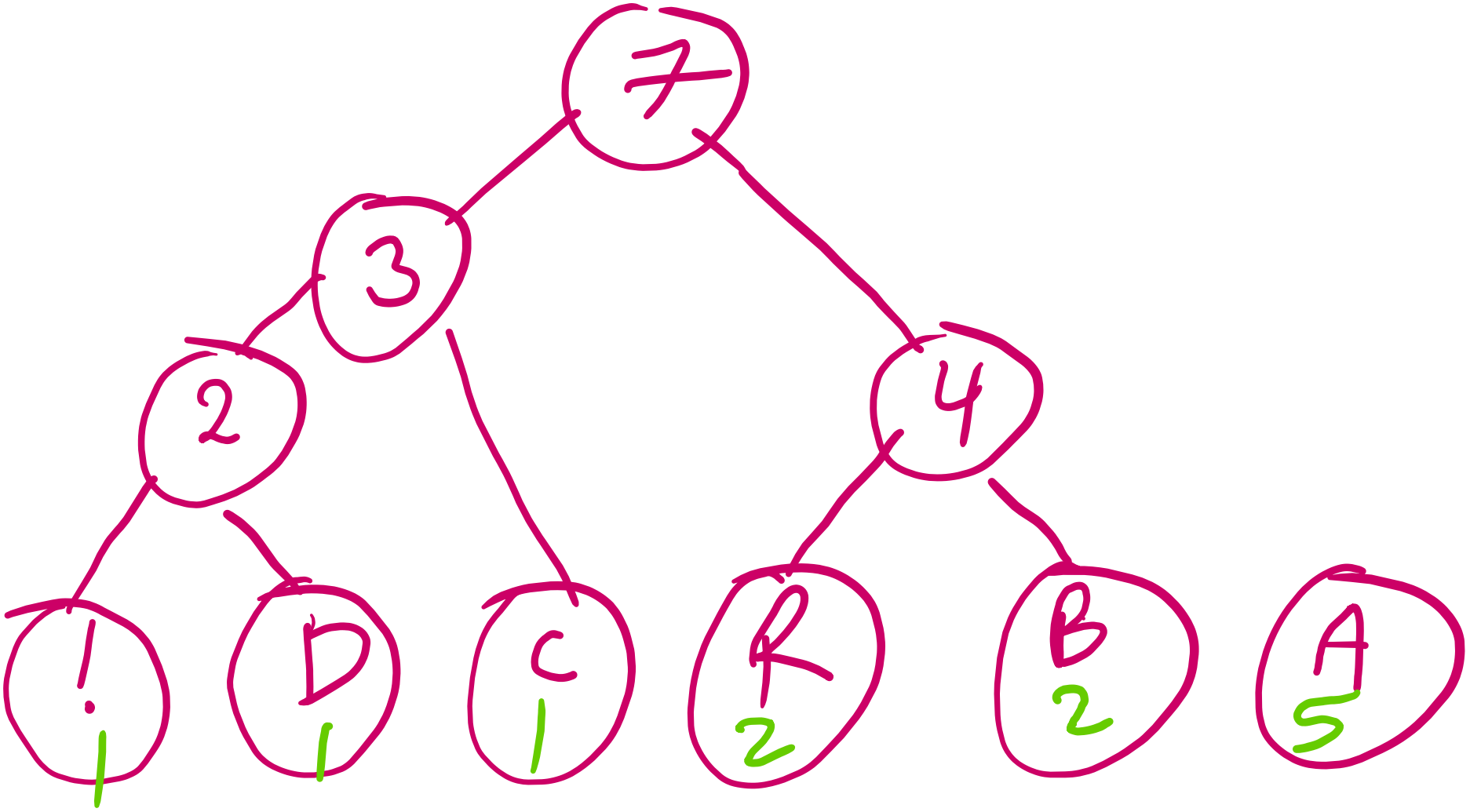
Example

- Build a tree for “ABRACADABRA!”



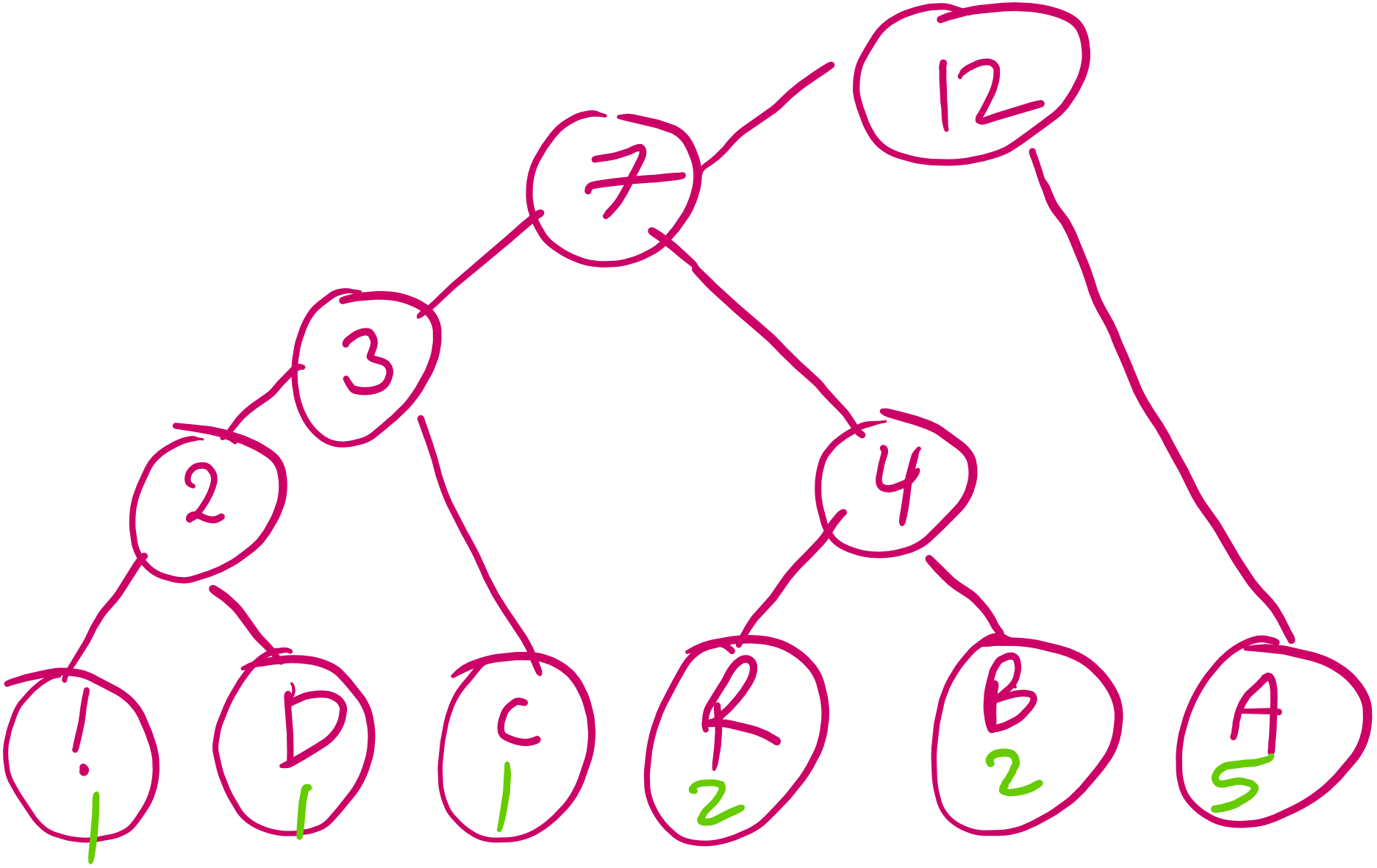
Example

- Build a tree for “ABRACADABRA!”



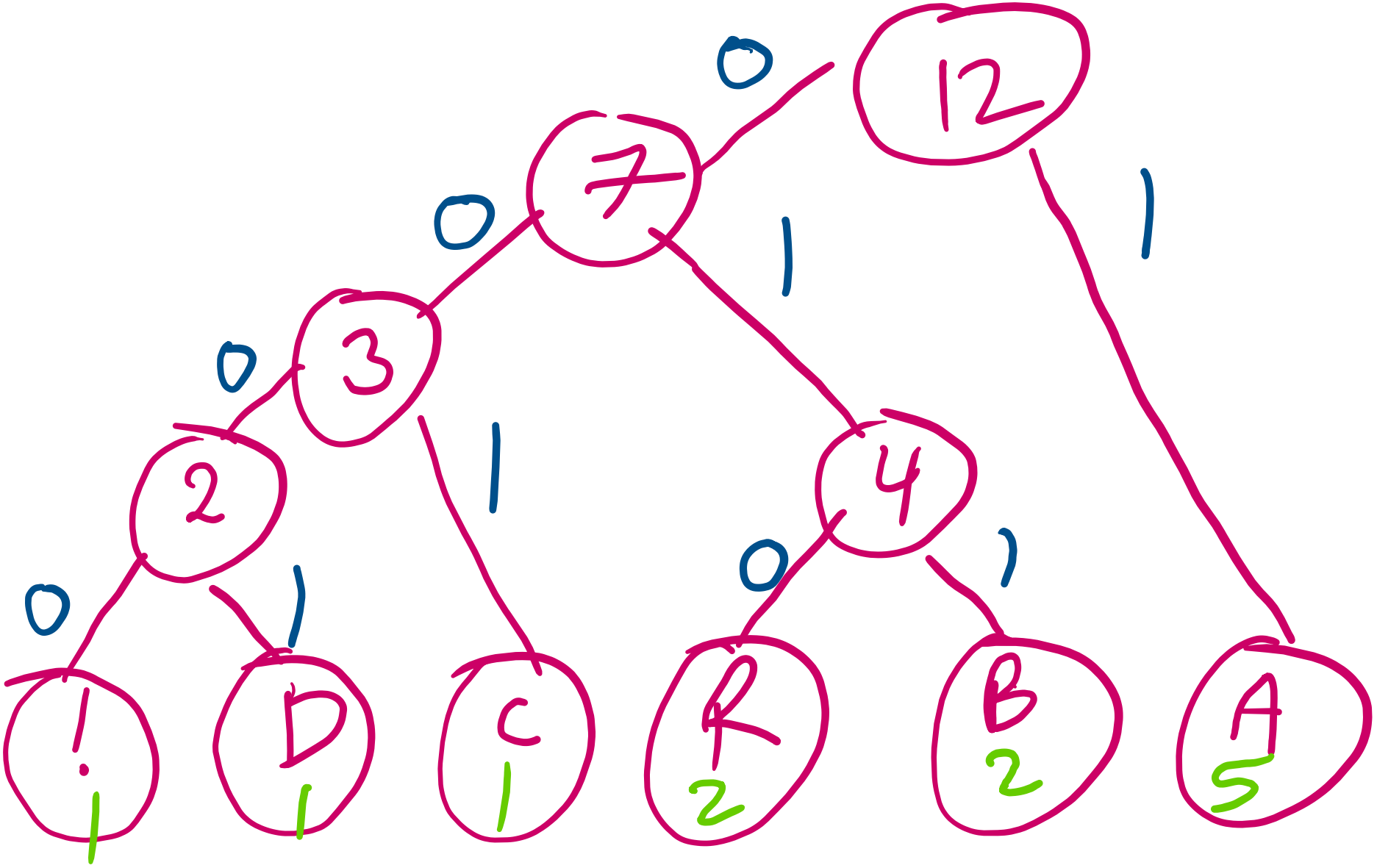
Example

- Build a tree for “ABRACADABRA!”



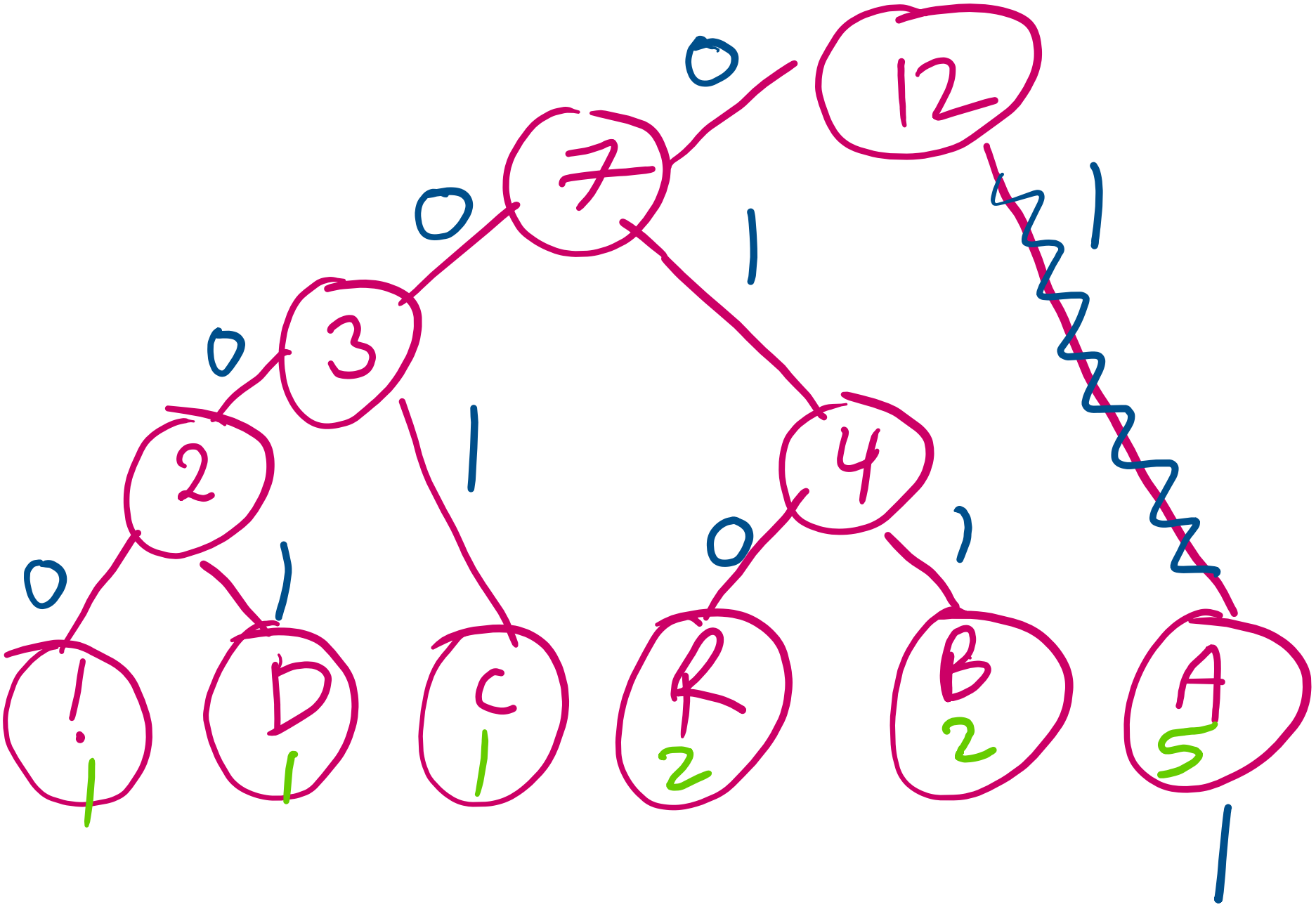
Example

- Build a tree for “ABRACADABRA!”



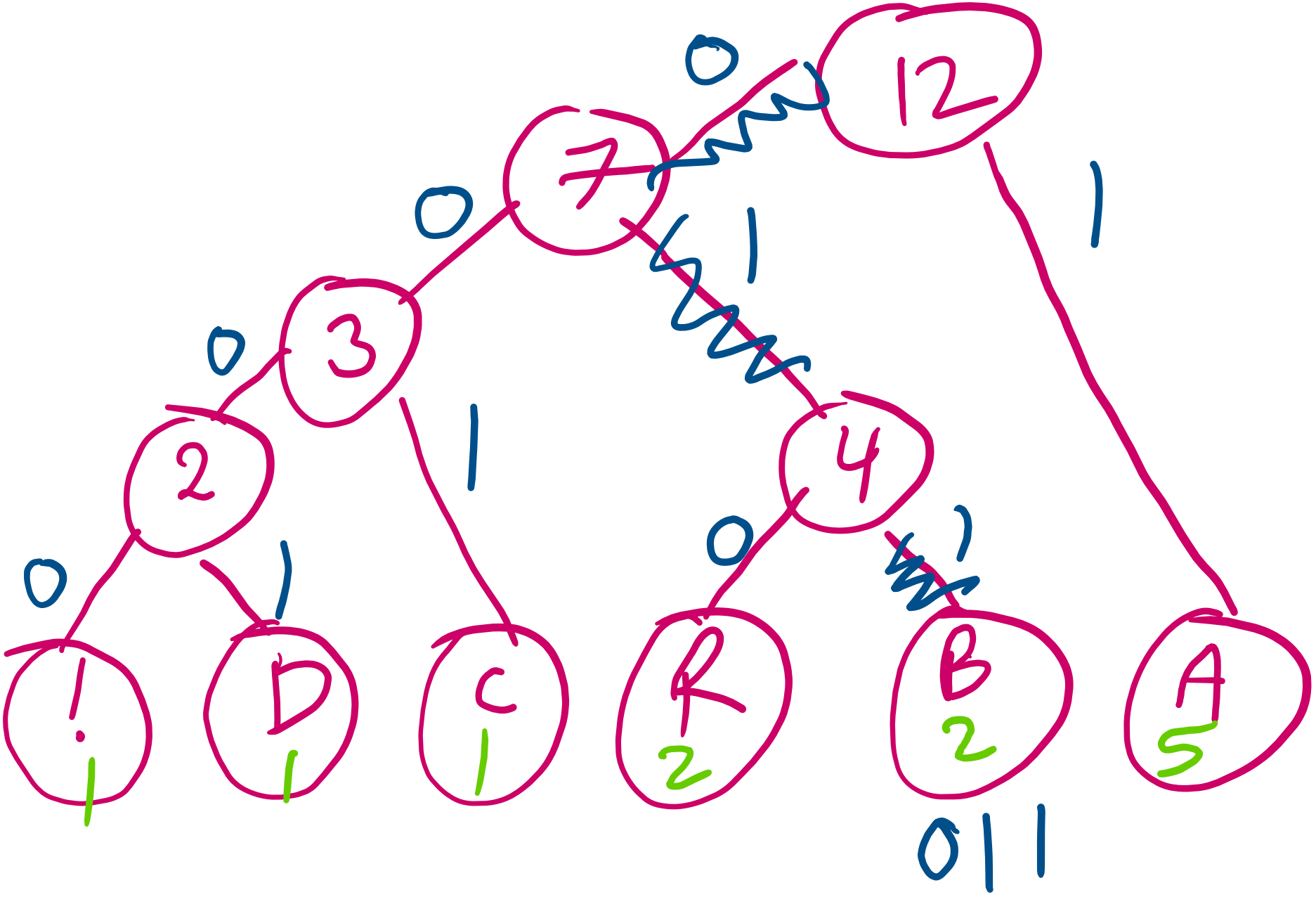
Example

- Build a tree for “ABRACADABRA!”



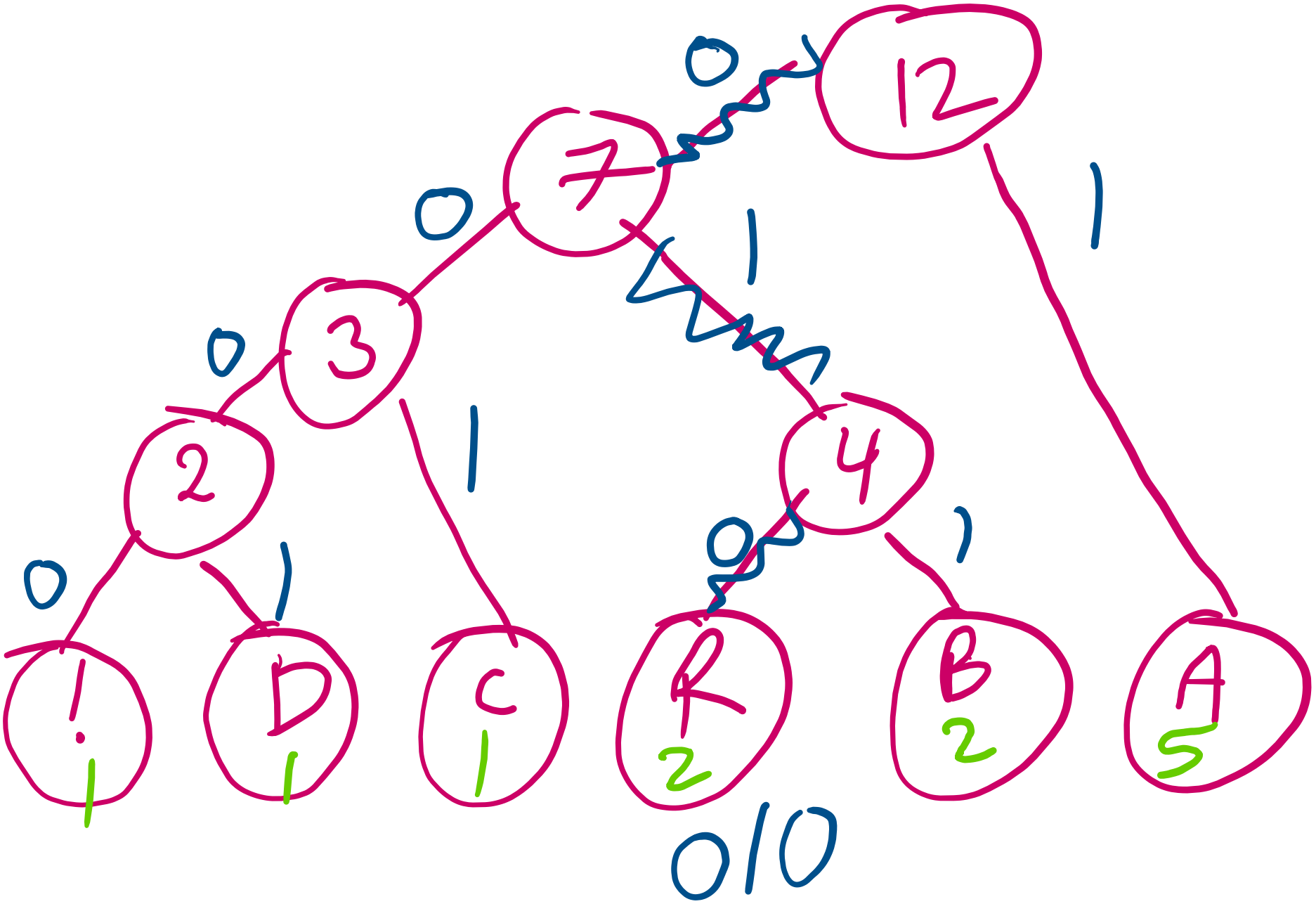
Example

- Build a tree for “ABRACADABRA!”



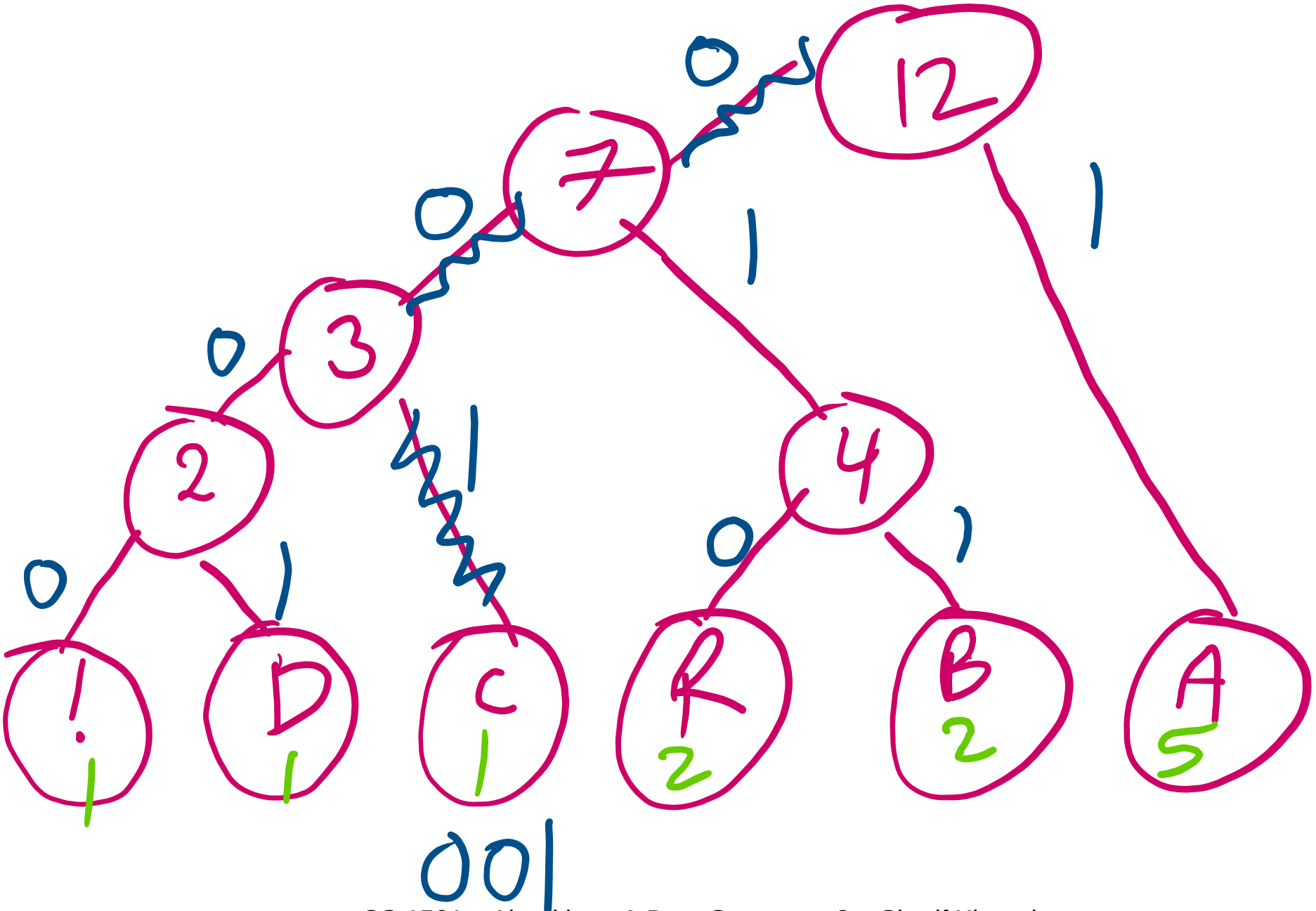
Example

- Build a tree for “ABRACADABRA!”



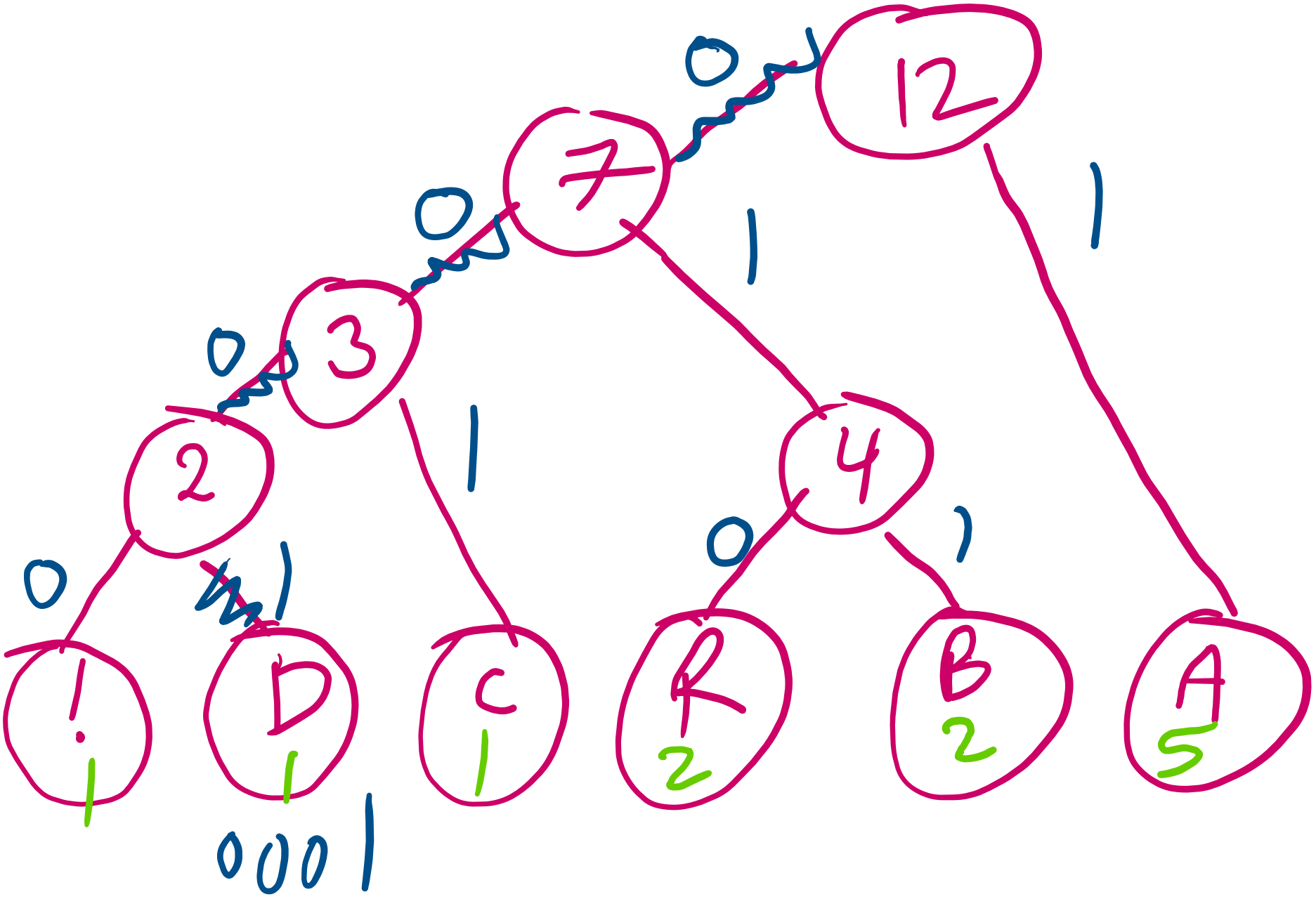
Example

- Build a tree for “ABRACADABRA!”



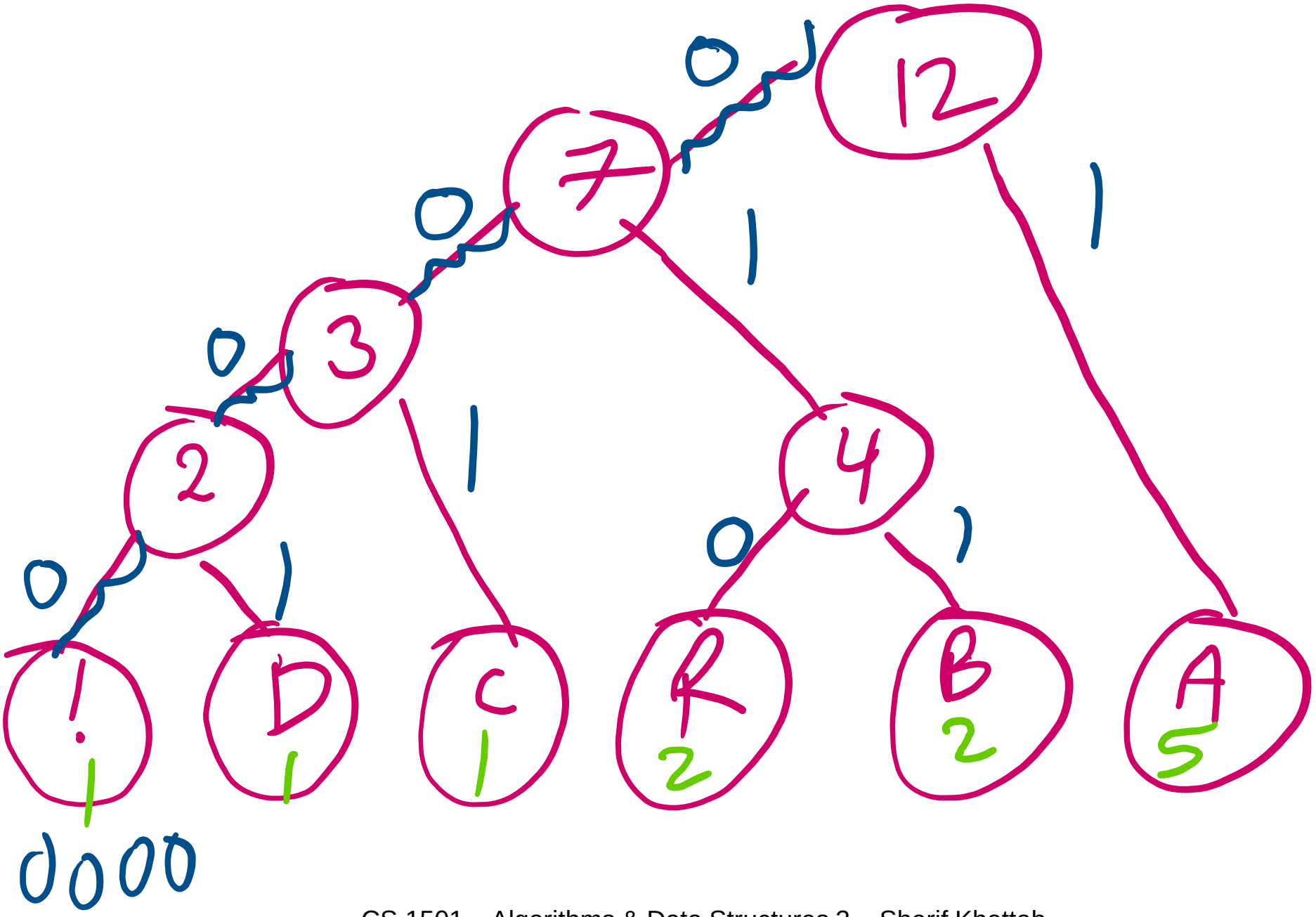
Example

- Build a tree for “ABRACADABRA!”



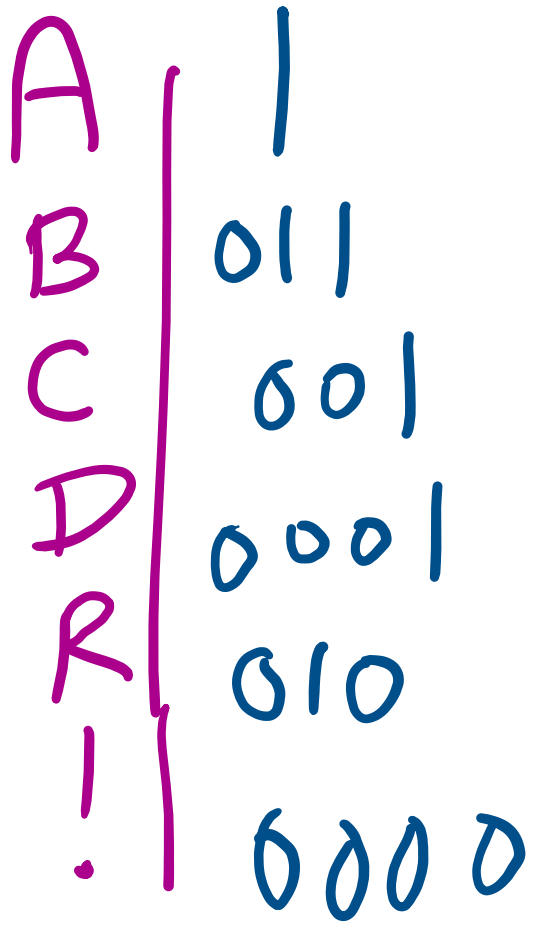
Example

- Build a tree for “ABRACADABRA!”



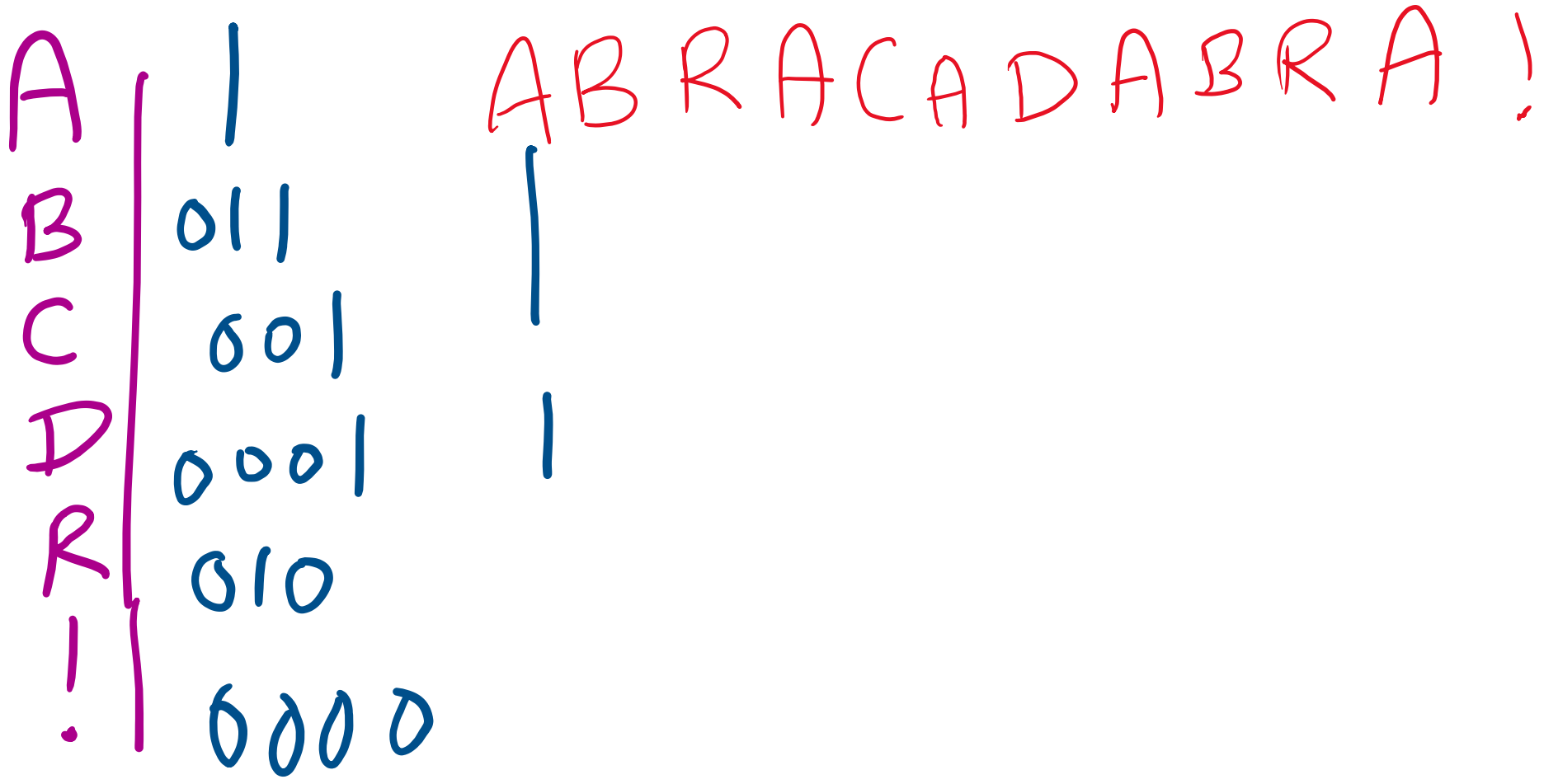
Example

- Build a tree for “ABRACADABRA!”



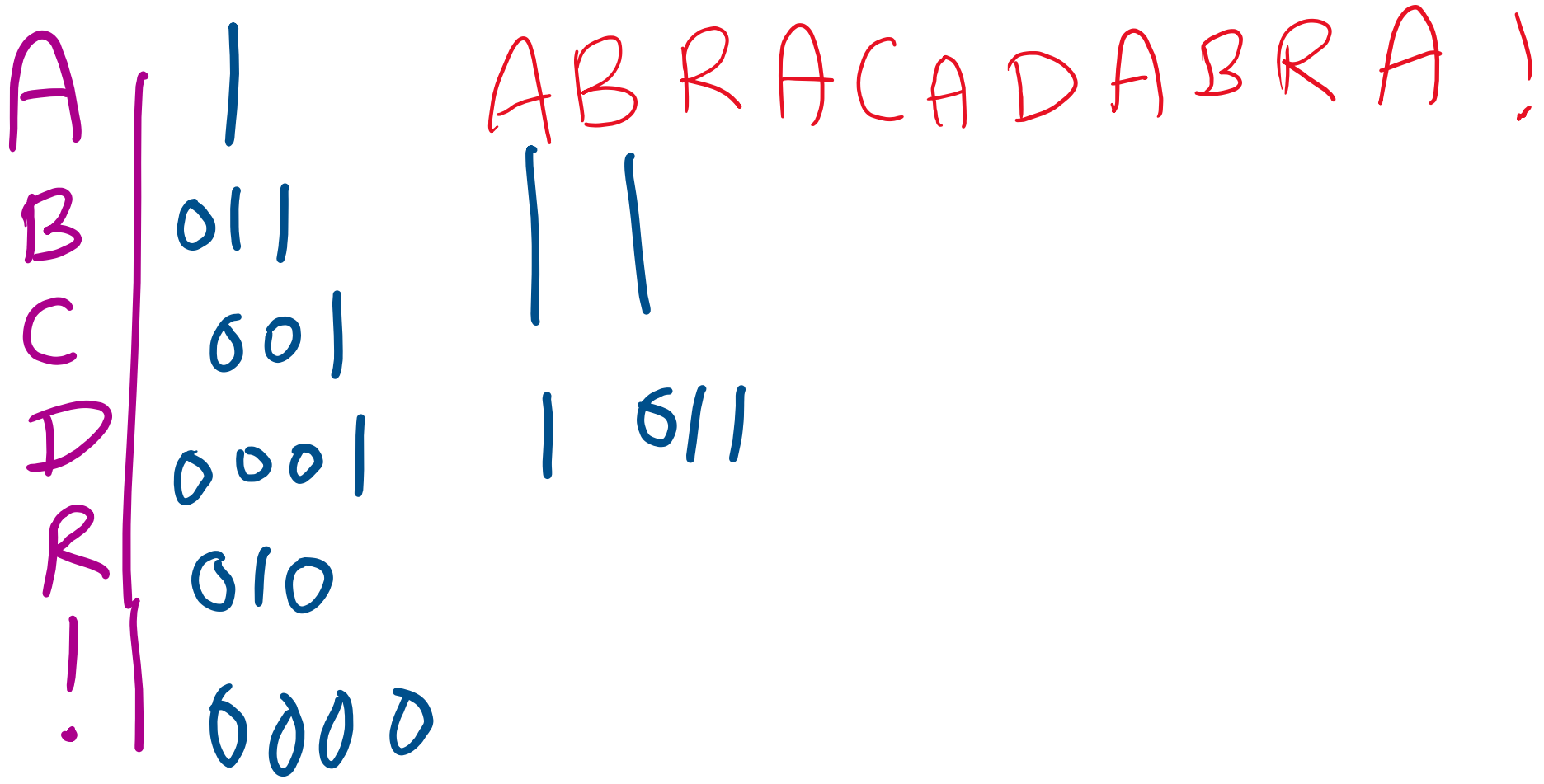
Example

- Build a tree for “ABRACADABRA!”



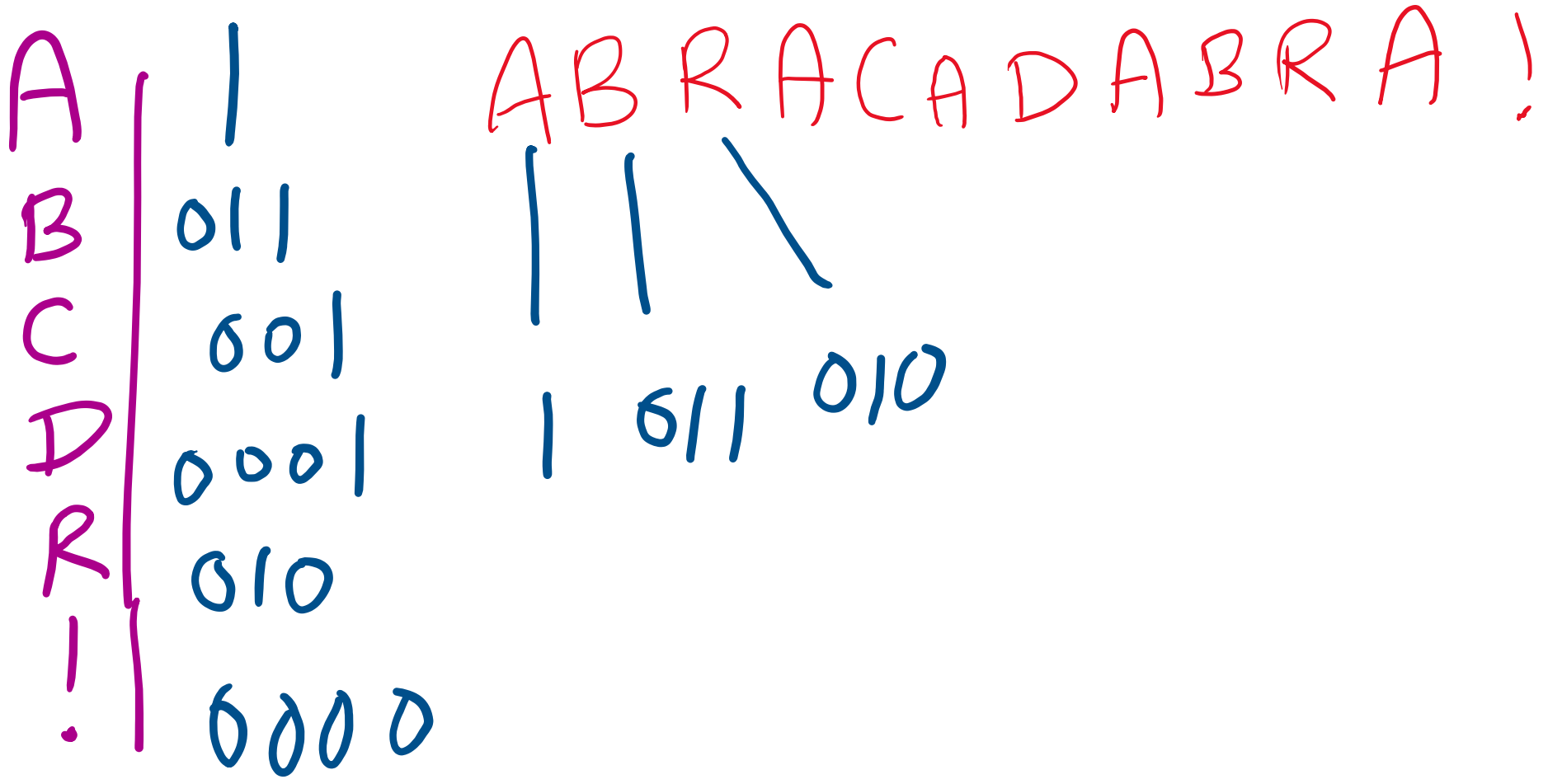
Example

- Build a tree for “ABRACADABRA!”



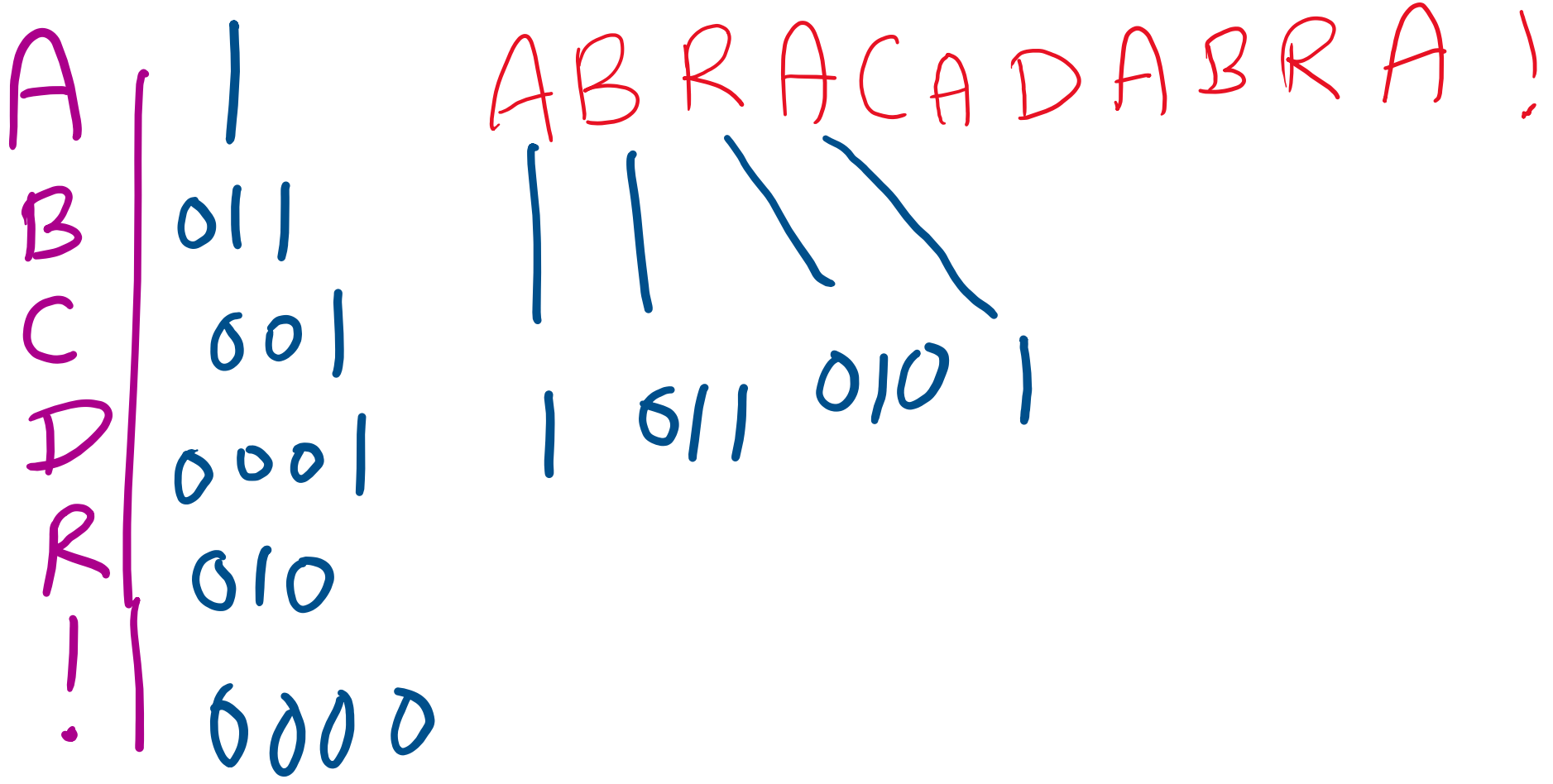
Example

- Build a tree for “ABRACADABRA!”



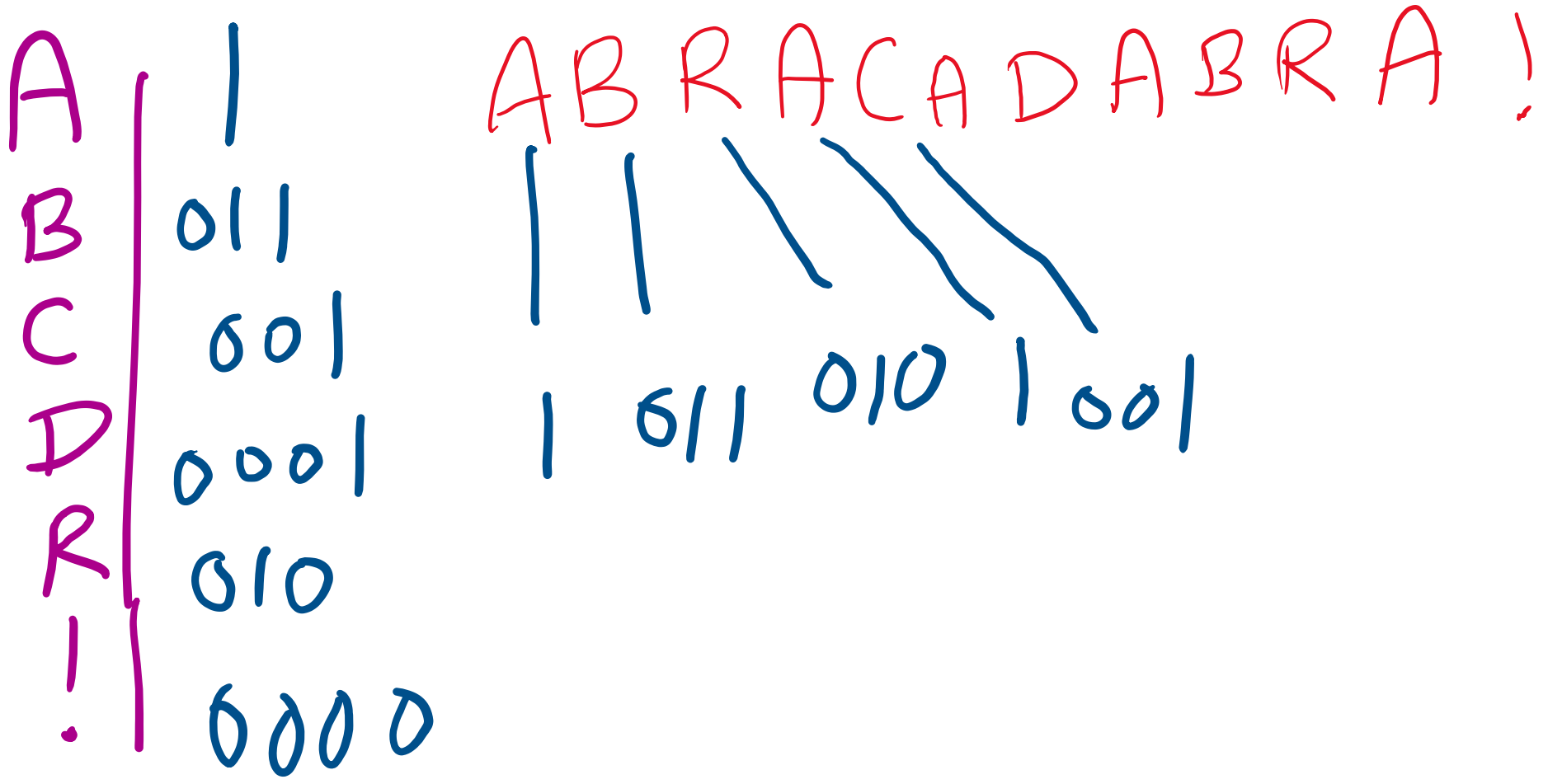
Example

- Build a tree for “ABRACADABRA!”



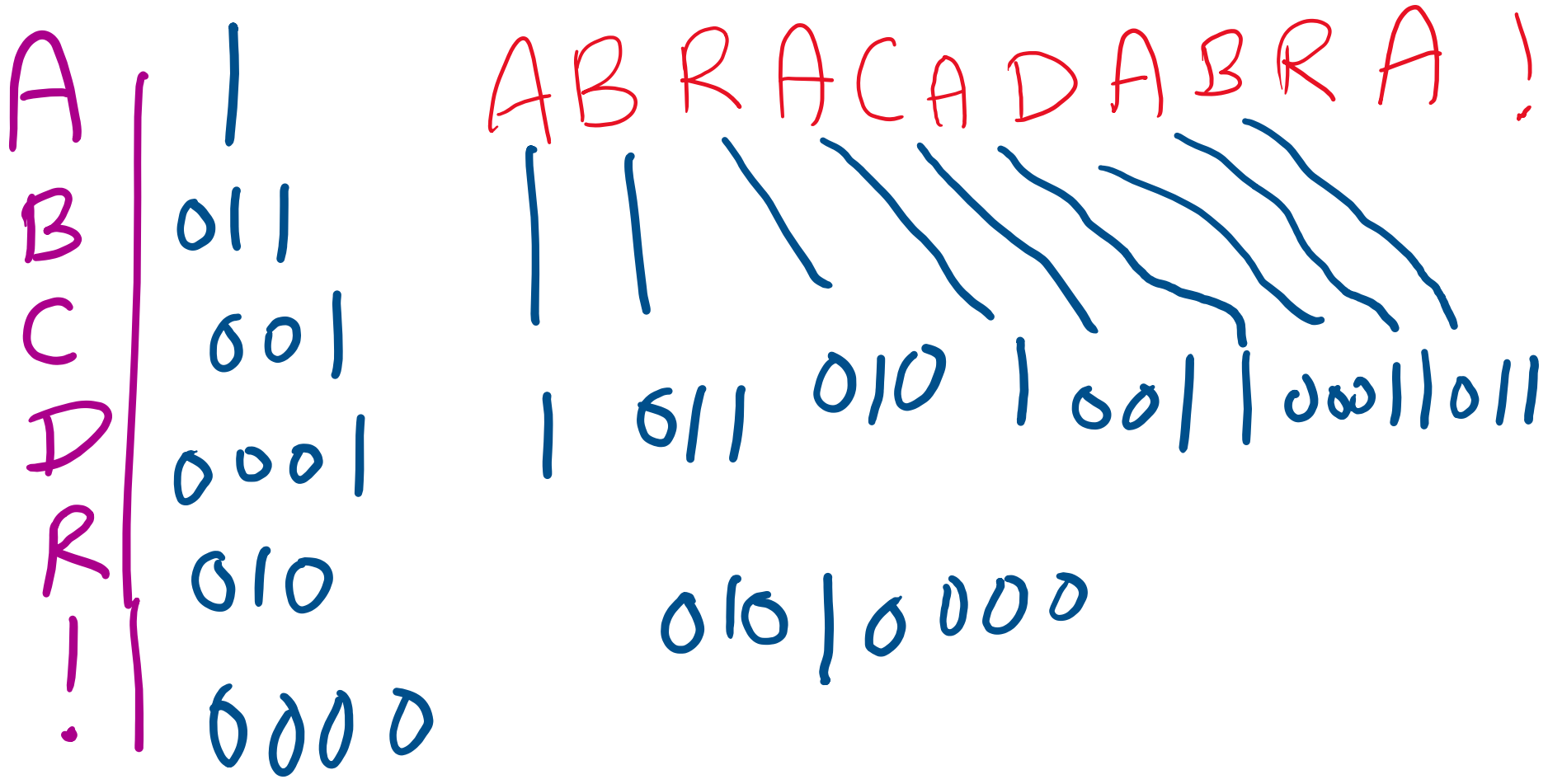
Example

- Build a tree for “ABRACADABRA!”



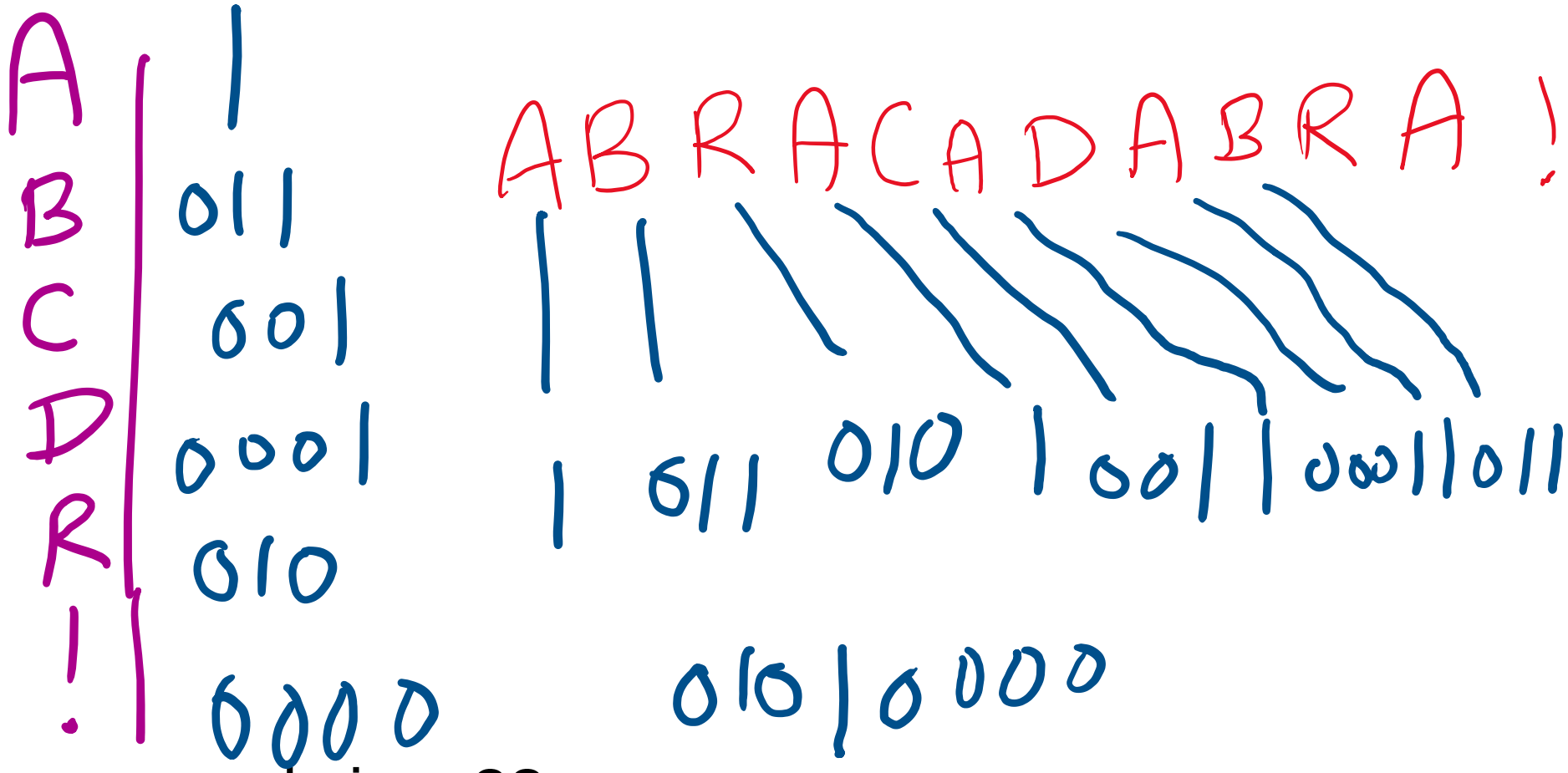
Example

- Build a tree for “ABRACADABRA!”



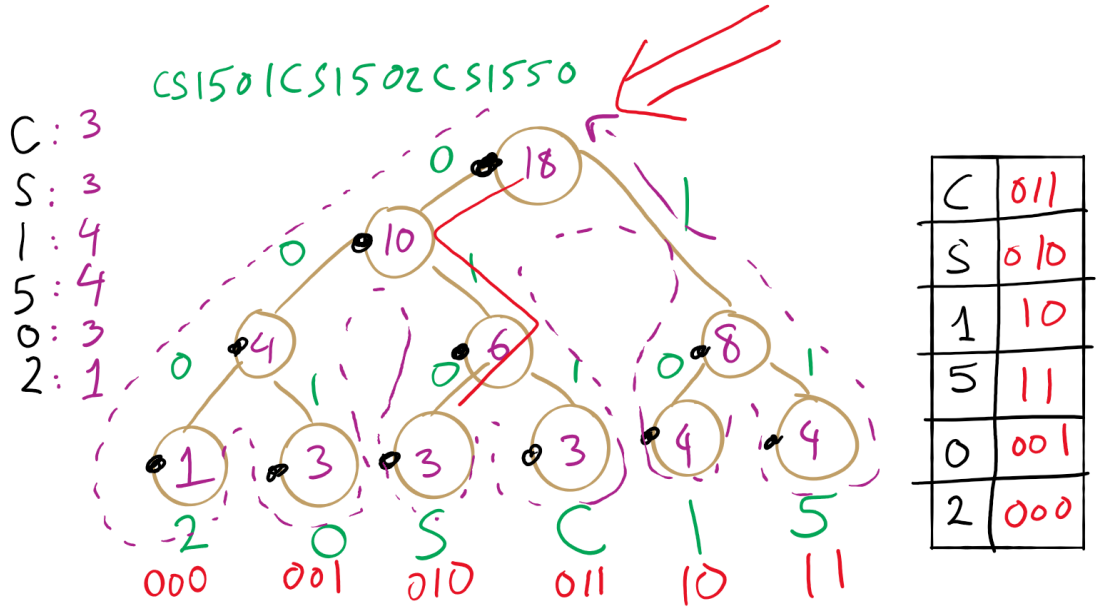
Example

- Build a tree for “ABRACADABRA!”



- Compressed size: 28
- Original size: 12*8 = 96 bits
- Compression ratio = 28/96

Huffman Compression Example



C	011
S	010
1	10
5	11
0	001
2	000

C S 1 5 0 ! C S 1 5 0 2 C S 1 5 5 0

011 010 10 11 001 10 011 010 10 11 001 000 011 010 10 11 11 001

Trie

0001 [ASCII for 2] 1 [ASCII for 0] 1 [ASCII for S] - - — — —

CS1

Implementation concerns

- Need to efficiently be able to select lowest weight trees to merge when constructing the trie
  - Can accomplish this using a *priority queue*
- Need to be able to read/write bitstrings!
  - Unless we pick multiples of 8 bits for our codewords, we will need to read/write *fractions* of bytes for our codewords
    - We're not actually going to do I/O on fraction of bytes
    - We'll maintain a buffer of bytes and perform bit processing on this buffer
    - See BinaryStdIn.java and BinaryStdOut.java

We need to read and write individual bits: Binary I/O

```
private static void writeBit(boolean bit) {
      // add bit to buffer by shifting in a zero
      buffer <<= 1;
      if (bit) buffer |= 1;  //then turning it to one if needed
      // if buffer is full (8 bits), write out as a single byte
      N++;
      if (N == 8) clearBuffer();
}
```

```
writeBit(true);
writeBit(false);
writeBit(true);
writeBit(false);
writeBit(false);
writeBit(false);
writeBit(false);
writeBit(true);
```

buffer: **00000000**

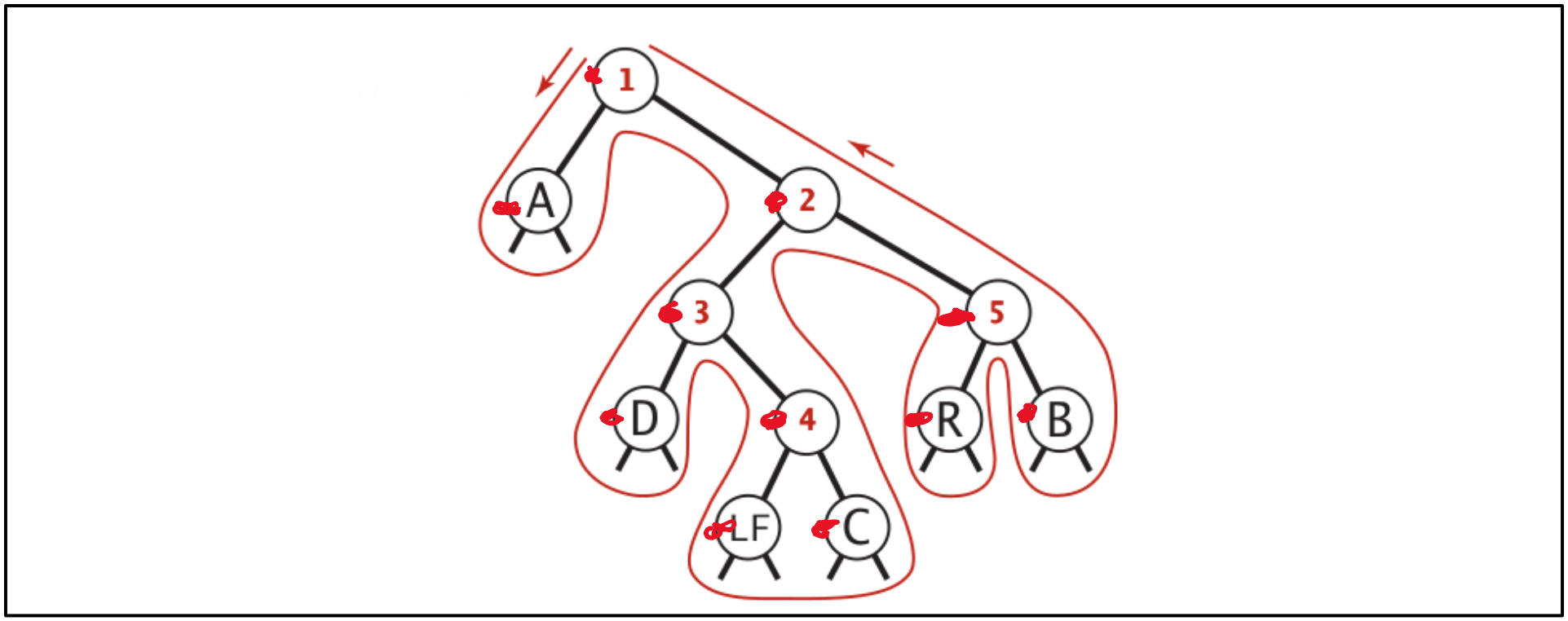
N: **0**

Further implementation concerns

- To encode, we'll need to read in characters and output codes
- To decode, we'll need to read in codes and output characters
- Sounds like we'll need a symbol table!
  - What implementation would be best?
    - Same for encoding and decoding?
  - Note that this means we need access to the trie to expand a compressed file!
    - Let's store the trie in the compressed file
    - how?

Representing tries as bitstrings

Preorder traversal

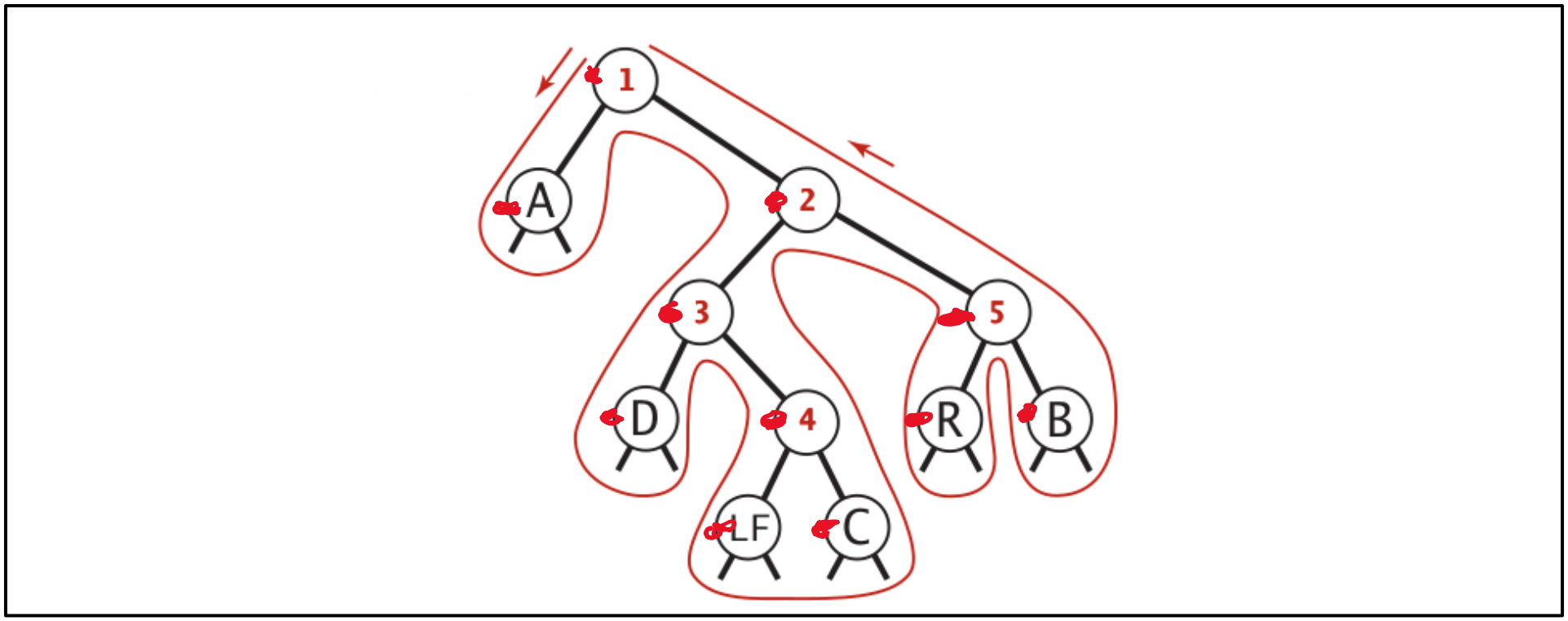


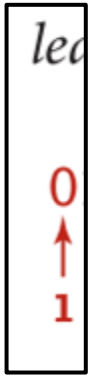
internal node → 0

leaf node → 1 followed by ASCII code of char inside

Representing tries as bitstrings

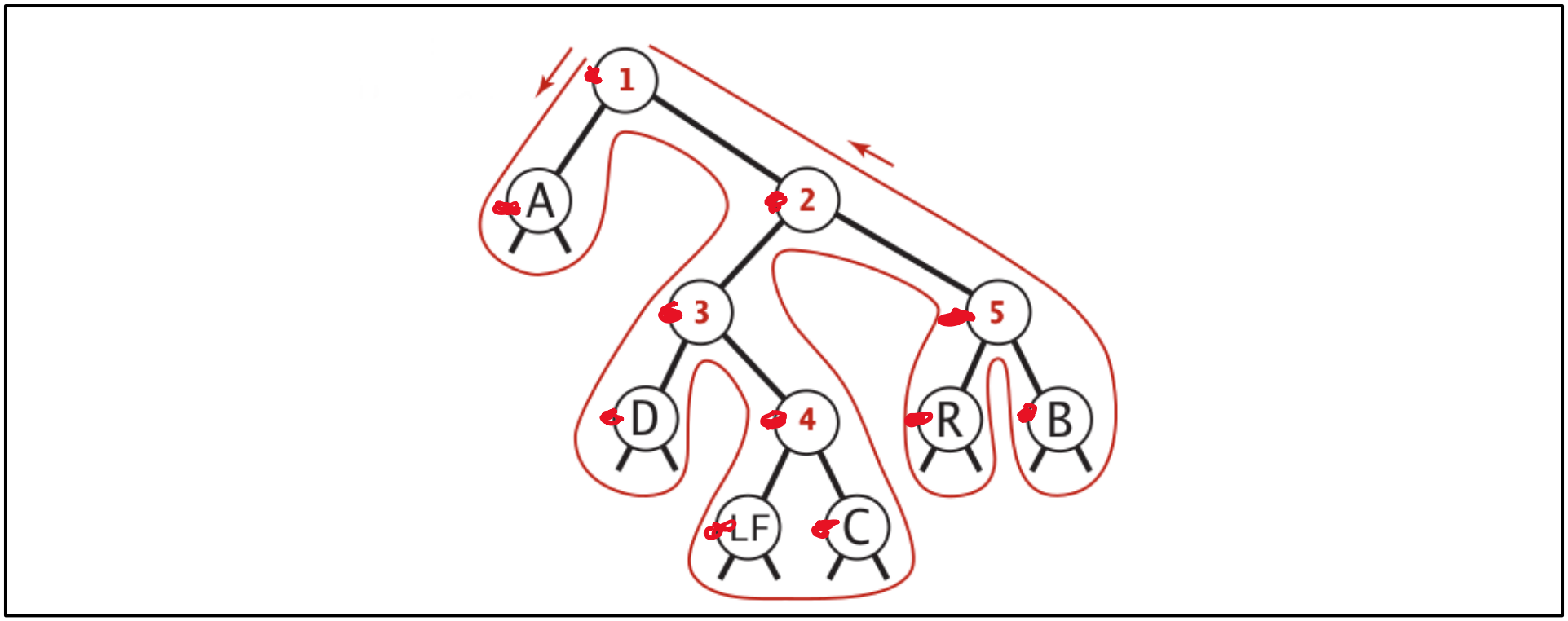
Preorder traversal

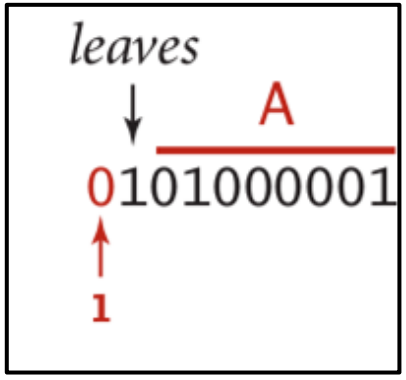




Representing tries as bitstrings

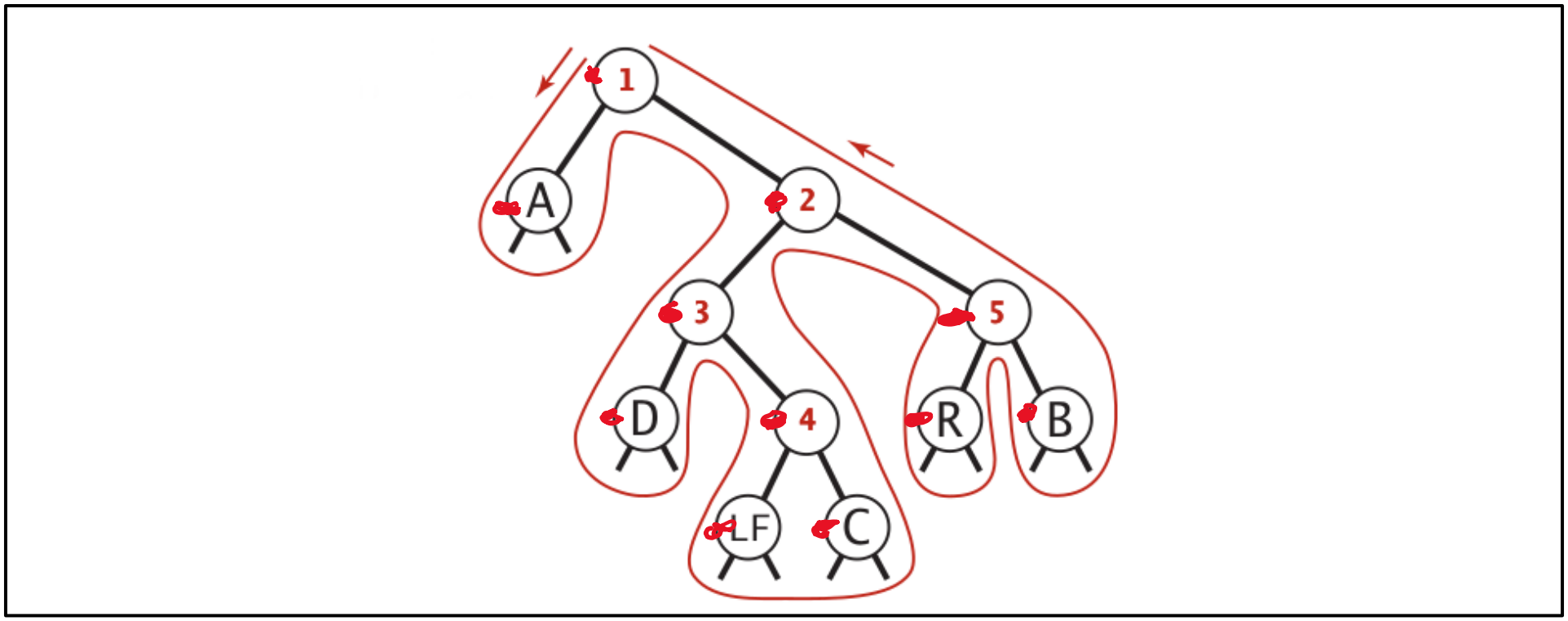
Preorder traversal

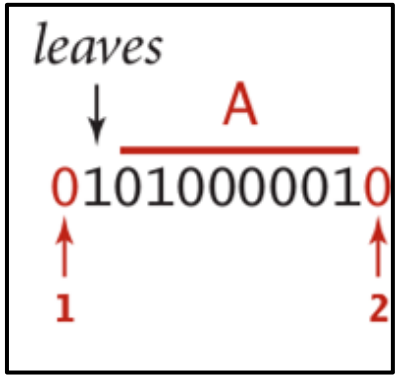




Representing tries as bitstrings

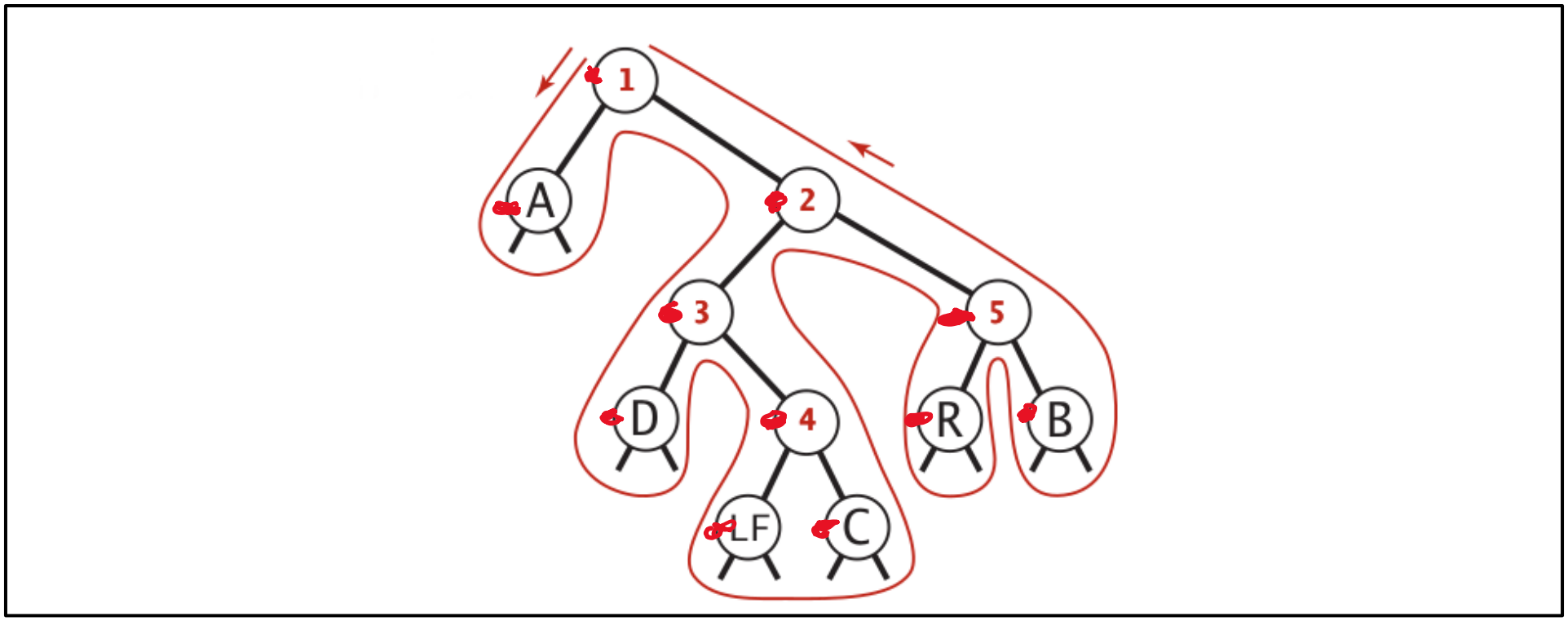
Preorder traversal

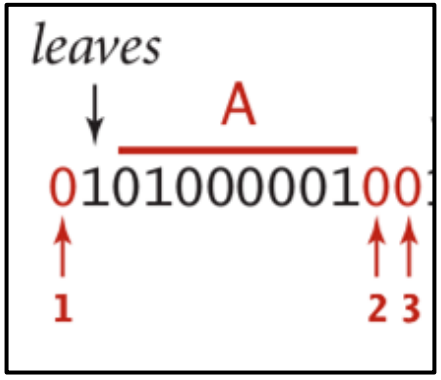




Representing tries as bitstrings

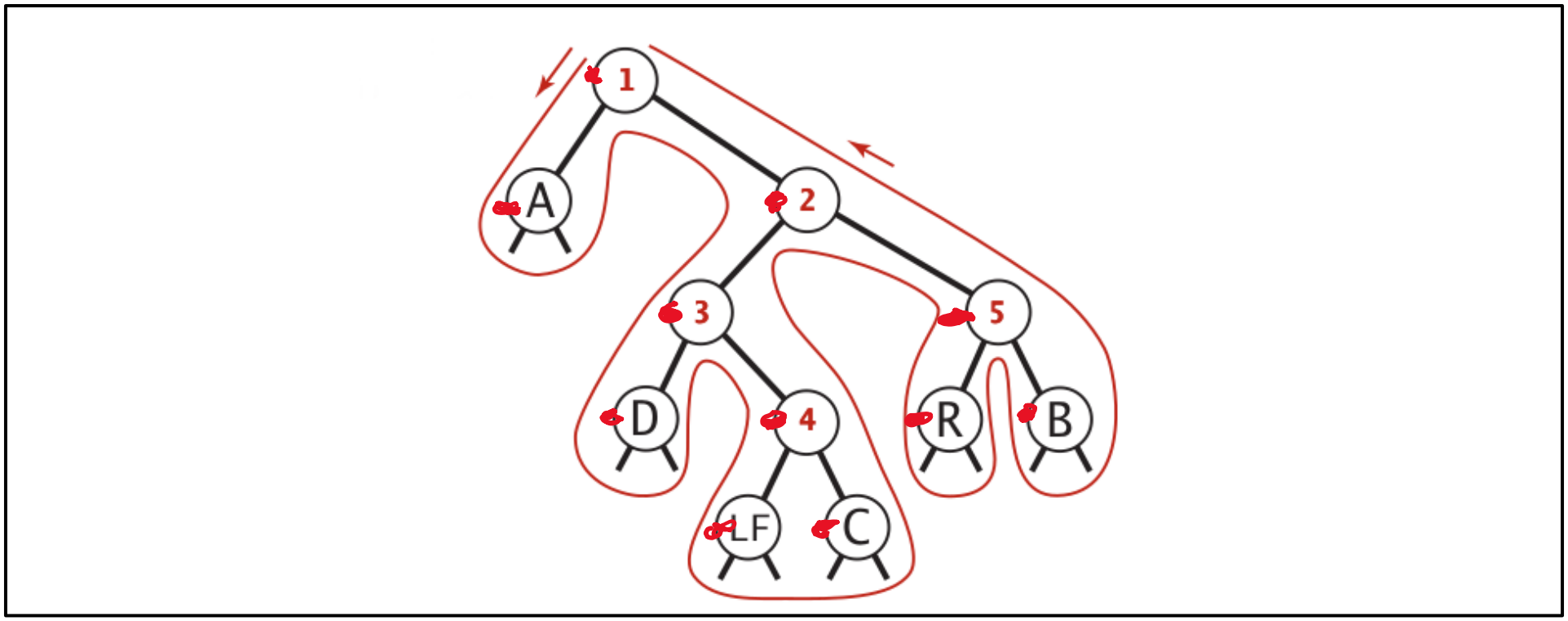
Preorder traversal

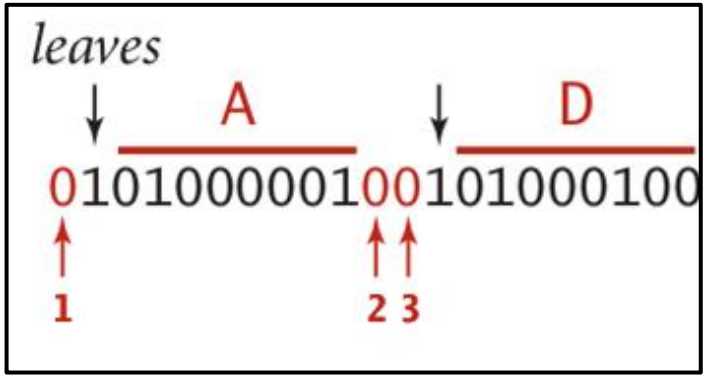




Representing tries as bitstrings

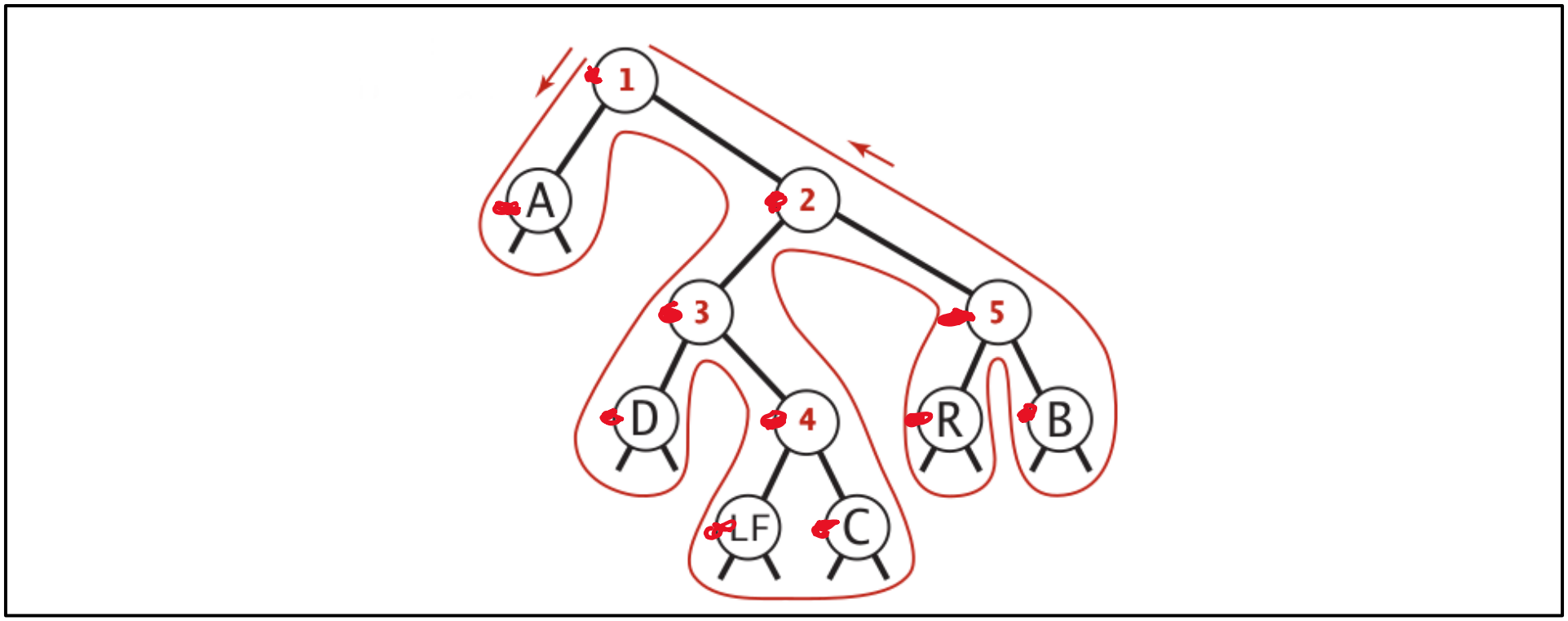
Preorder traversal

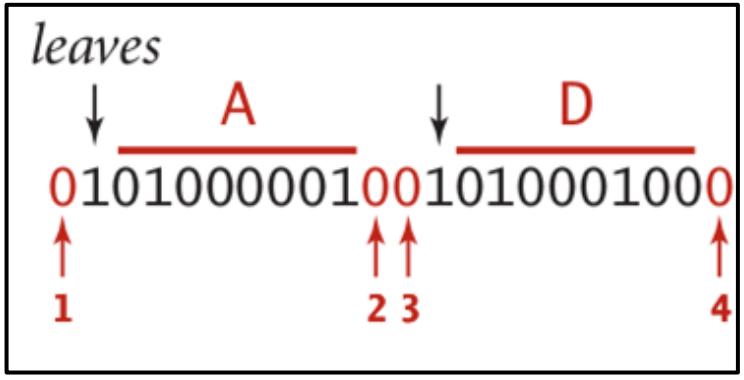




Representing tries as bitstrings

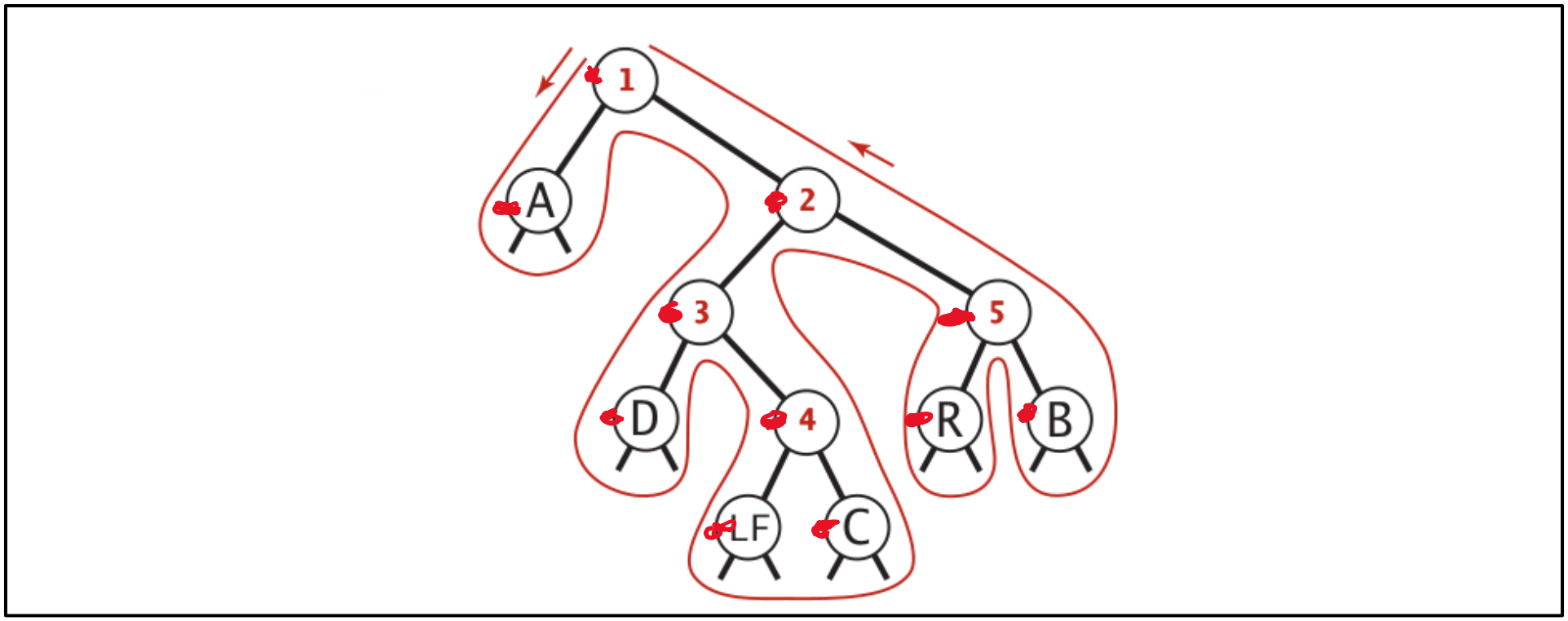
Preorder traversal

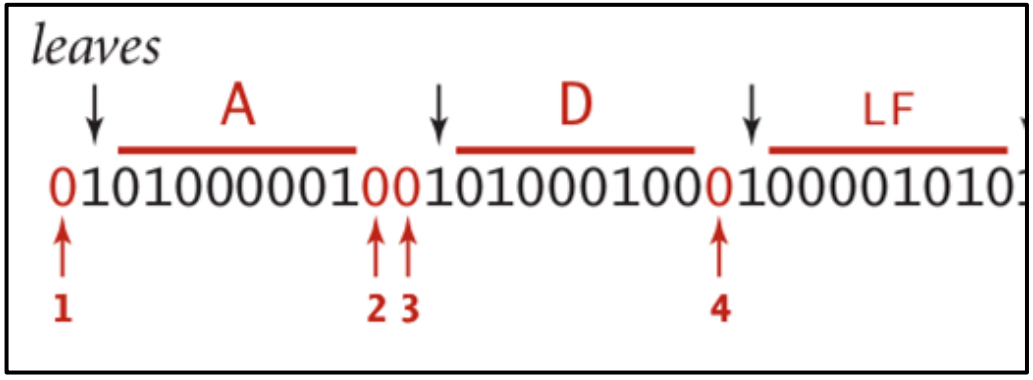




Representing tries as bitstrings

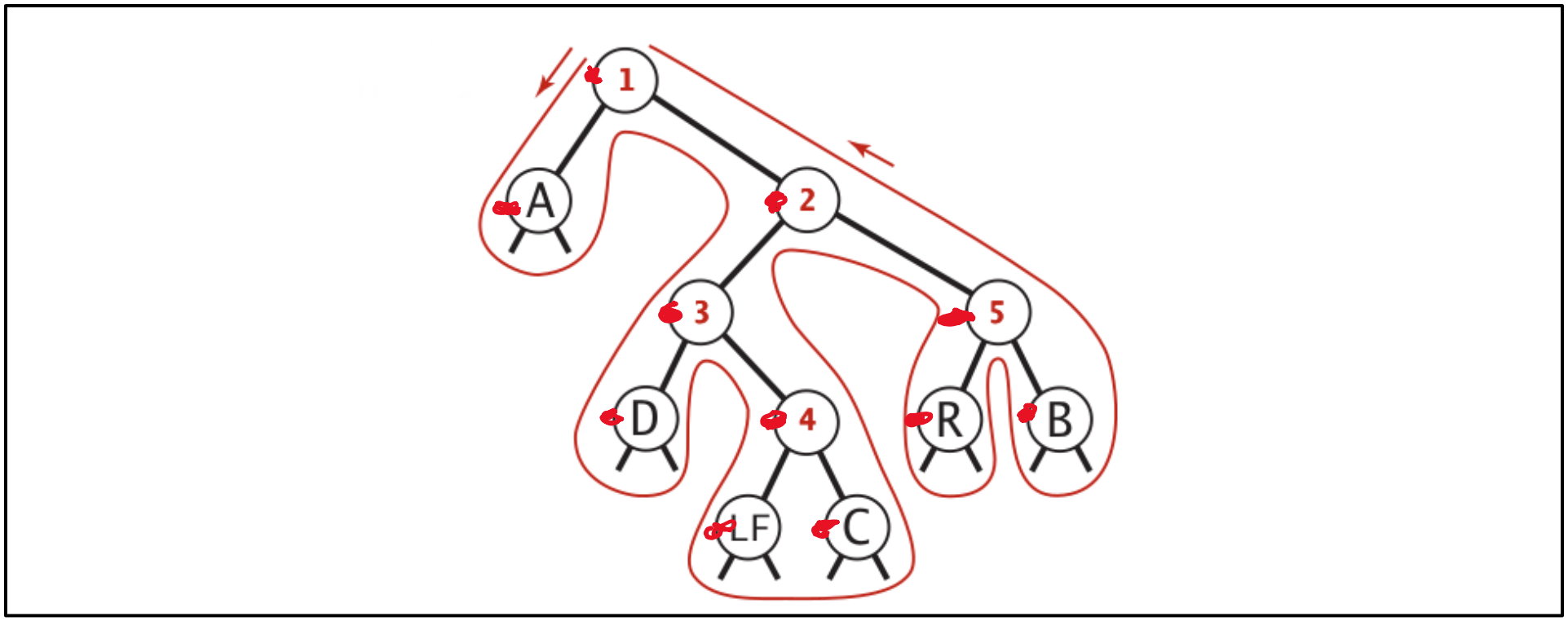
Preorder traversal

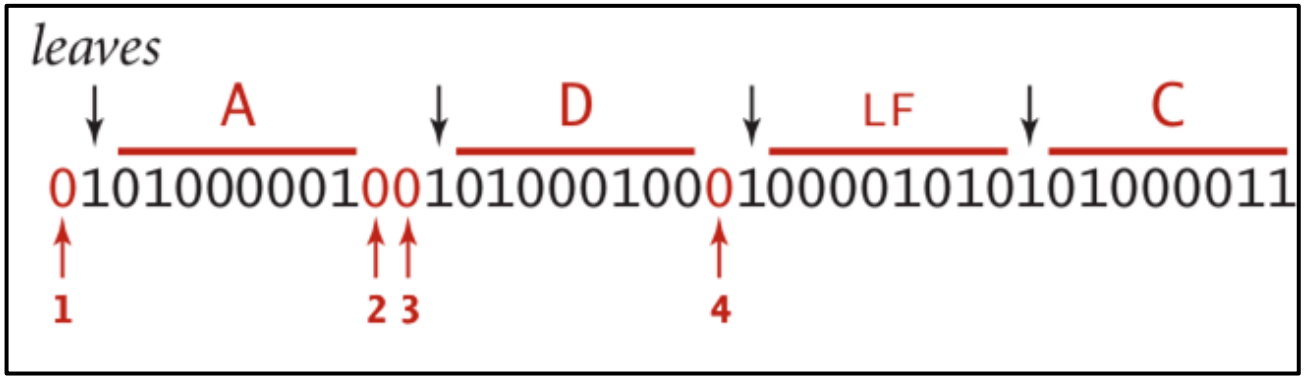




Representing tries as bitstrings

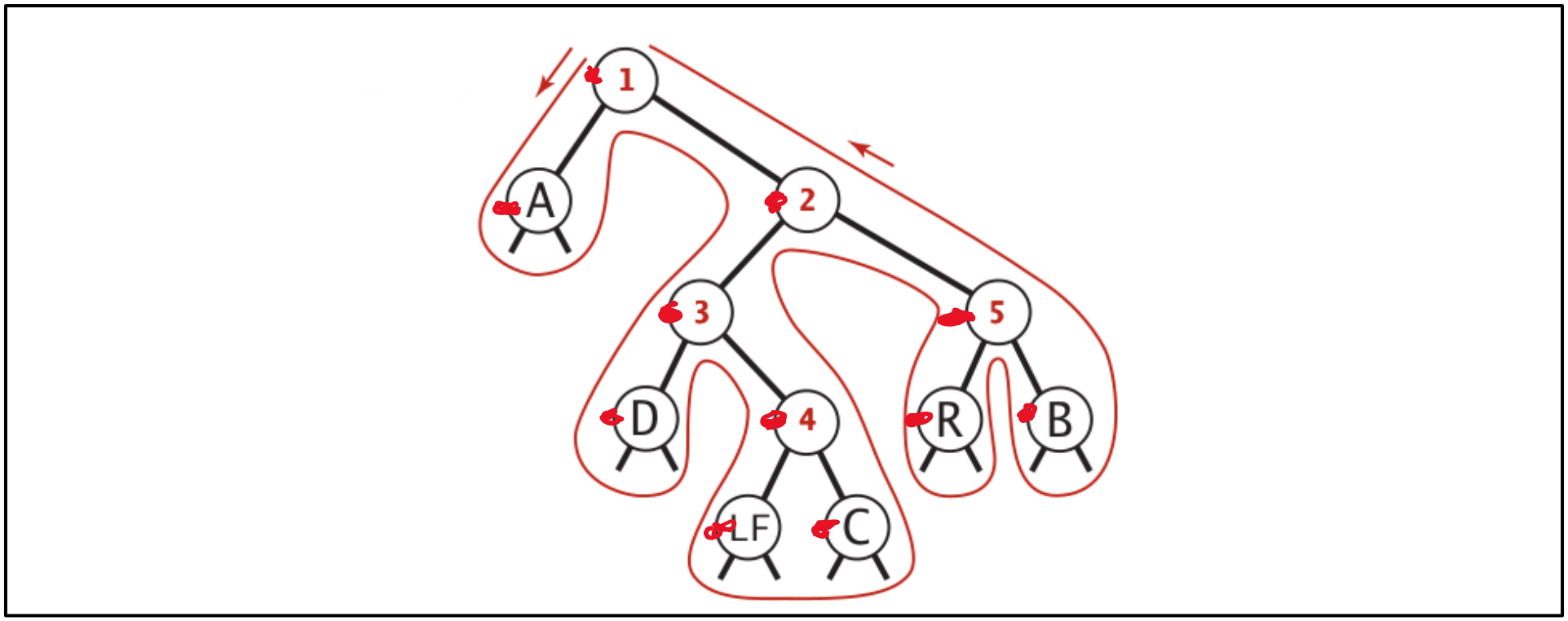
Preorder traversal

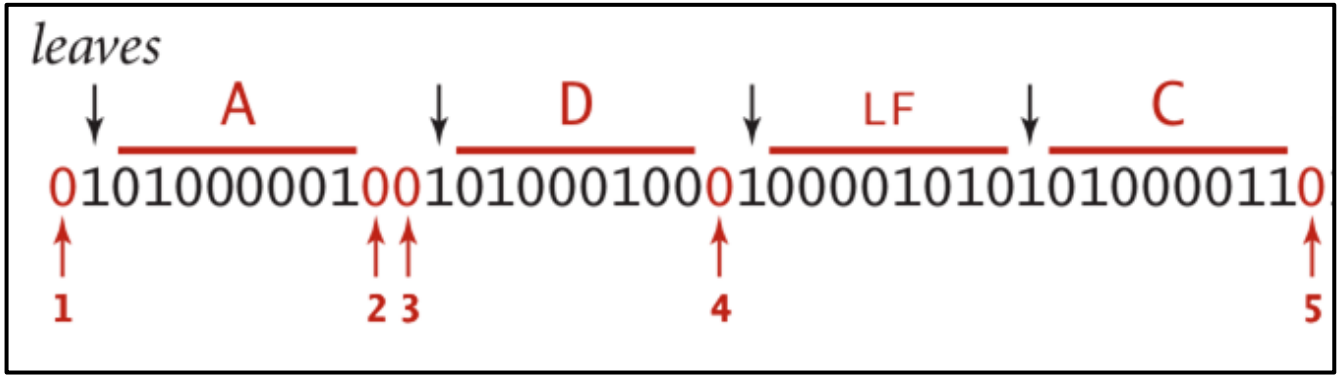




Representing tries as bitstrings

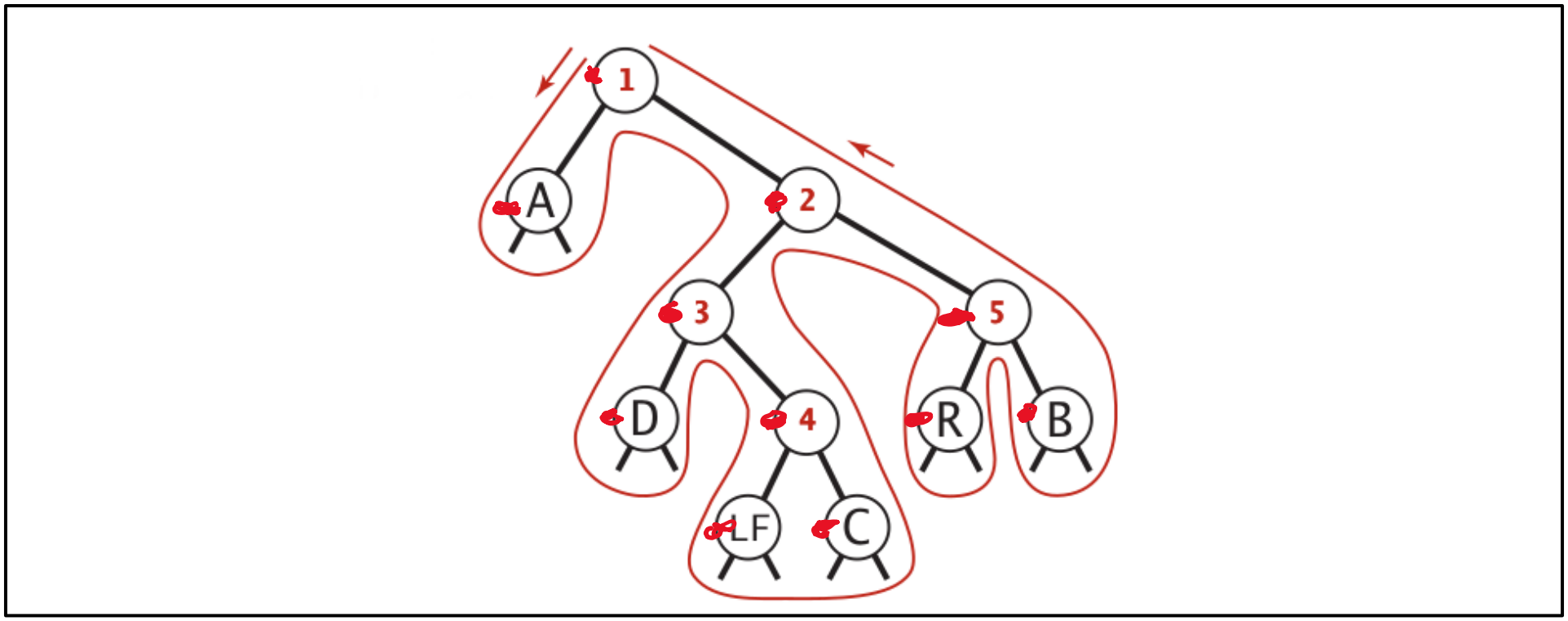
Preorder traversal

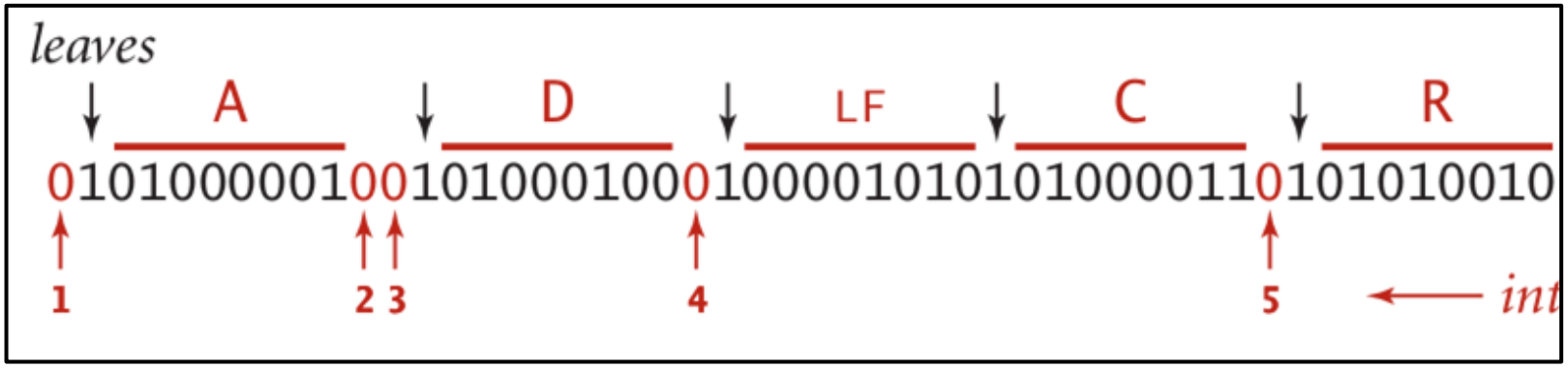




Representing tries as bitstrings

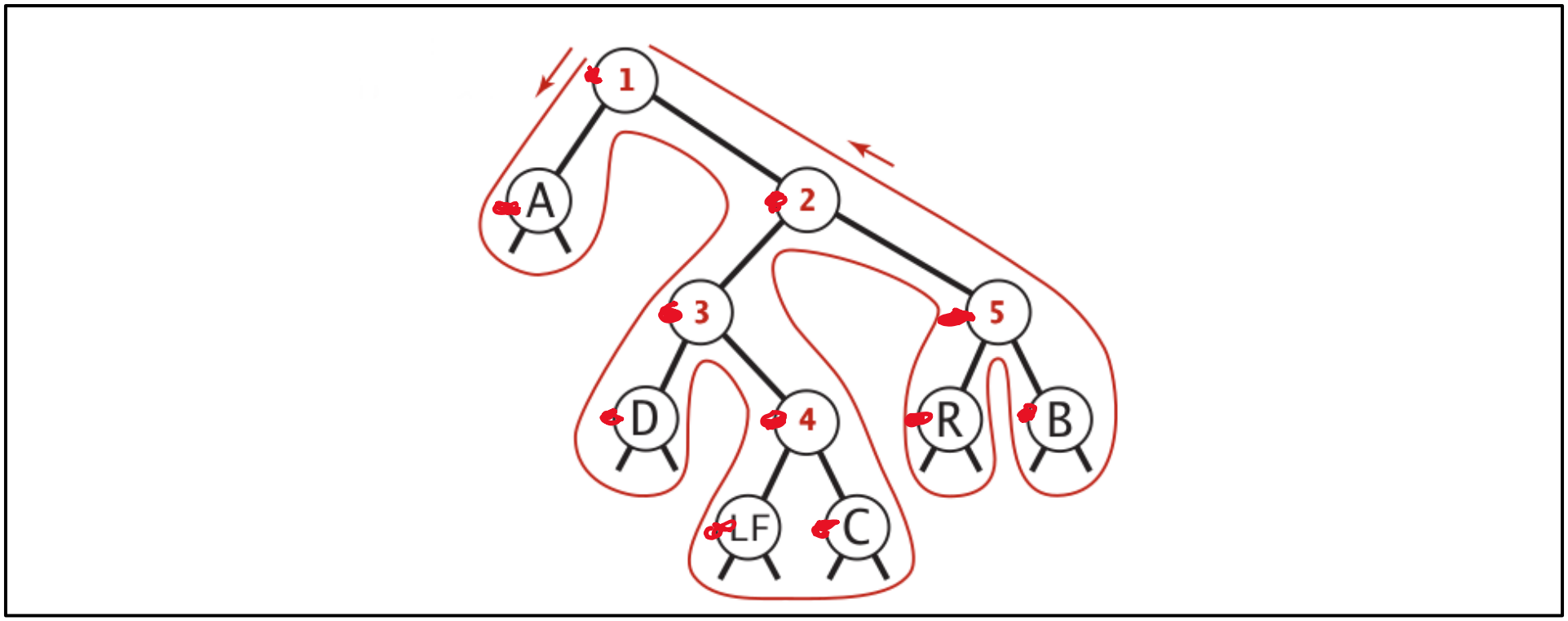
Preorder traversal

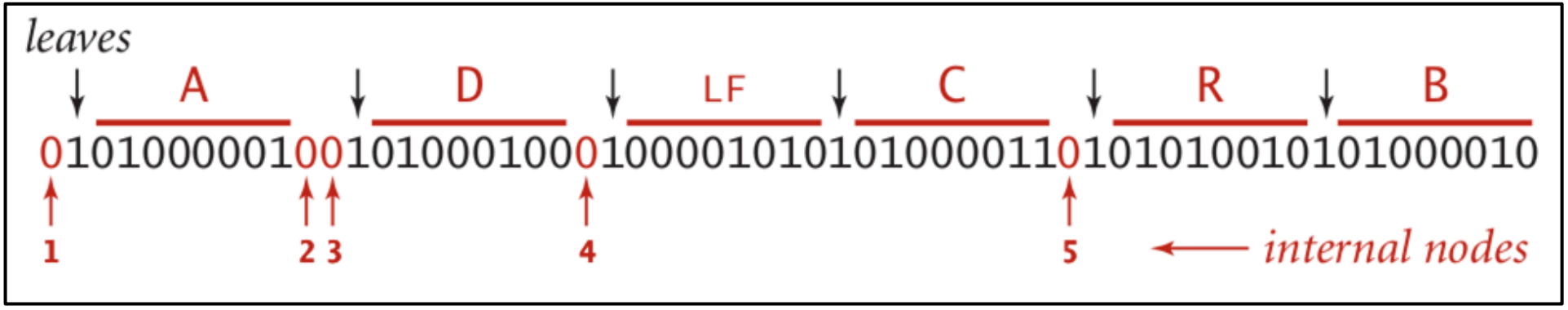




Representing tries as bitstrings

Preorder traversal





Writing a trie

```
private static void writeTrie(Node x){
   if (x.isLeaf()) {
      BinaryStdOut.write(true);
      BinaryStdOut.write(x.ch);
      return;
   }
   BinaryStdOut.write(false);
   writeTrie(x.left);
   writeTrie(x.right);
}
```

Reading a trie back

Similar to the read tree lab!

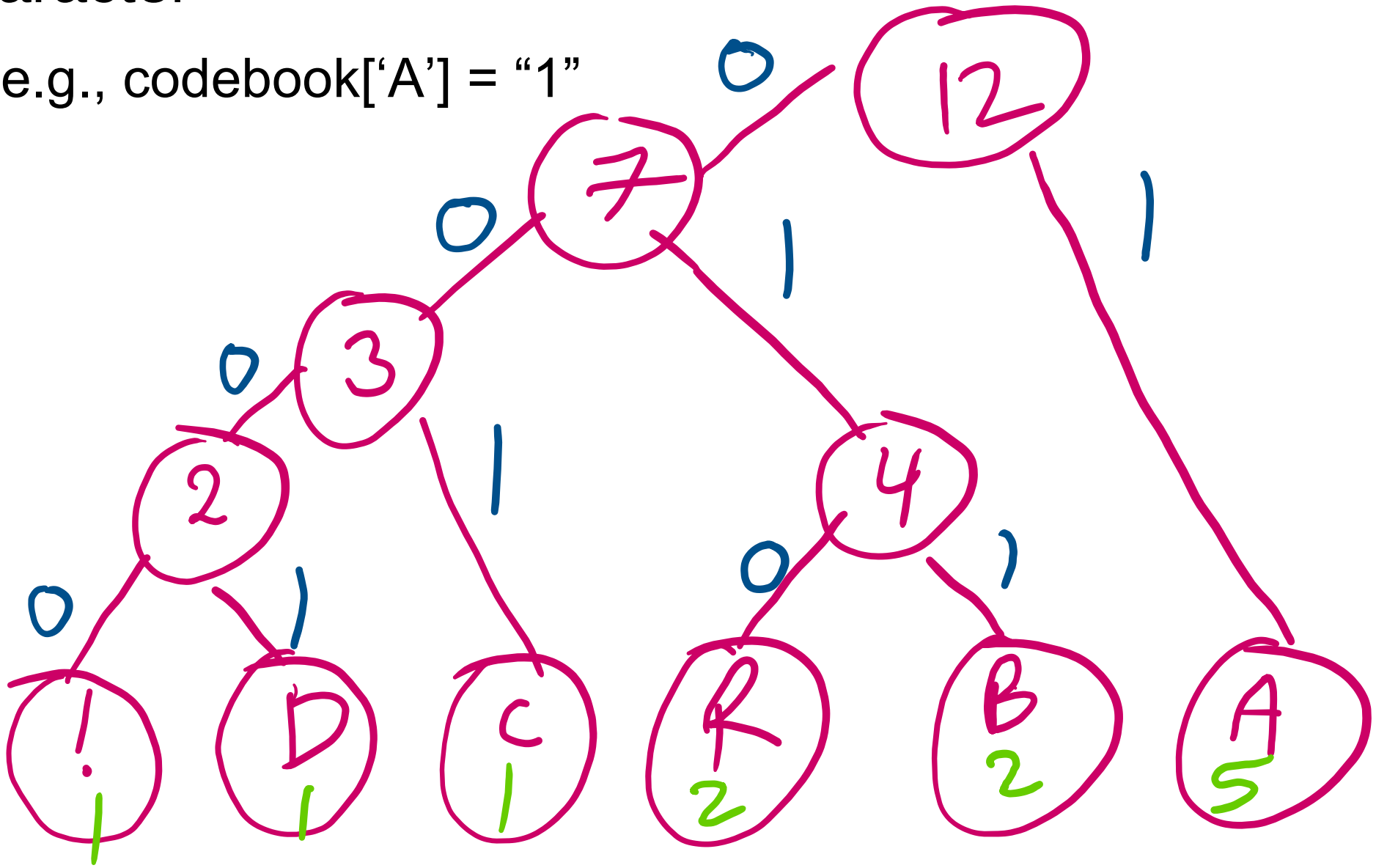
```
private static Node readTrie() {
  if (BinaryStdIn.readBoolean())
    return new
  Node(BinaryStdIn.readChar(), 0, null,
  null);
  return new Node('\0', 0, readTrie(),
readTrie());
}
```

Huffman pseudocode

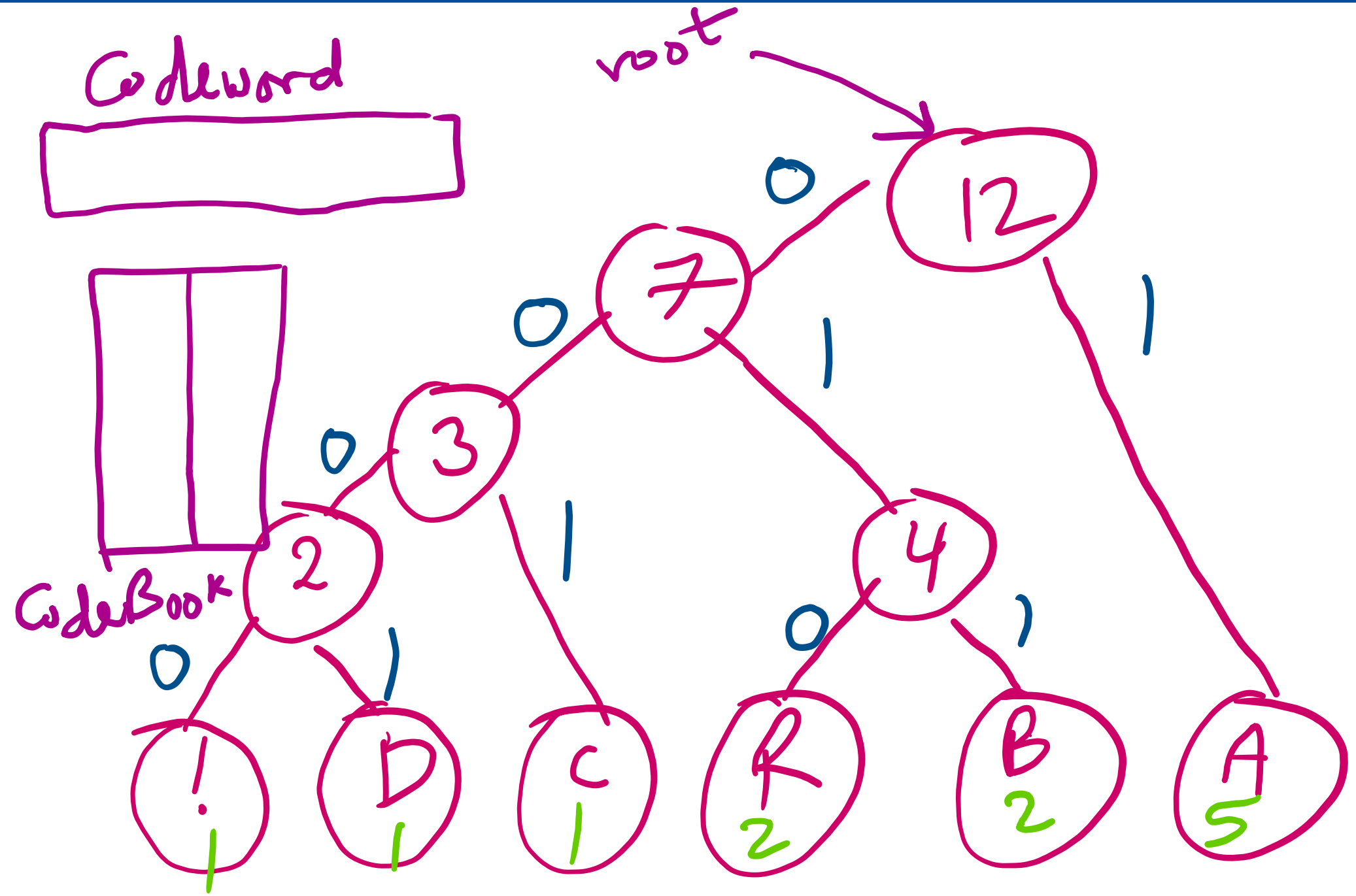
- Encoding approach:
  - Read input
  - Compute frequencies
  - Build trie/codeword table
  - Write out trie as a bitstring to compressed file
  - Write out character count of input (**why is that necessary**?)
  - Use table to write out the codeword for each input character
- Decoding approach:
  - Read trie
  - Read character count
  - Use trie to decode bitstring of compressed file

Huffman Compression: Generating the *Codebook*

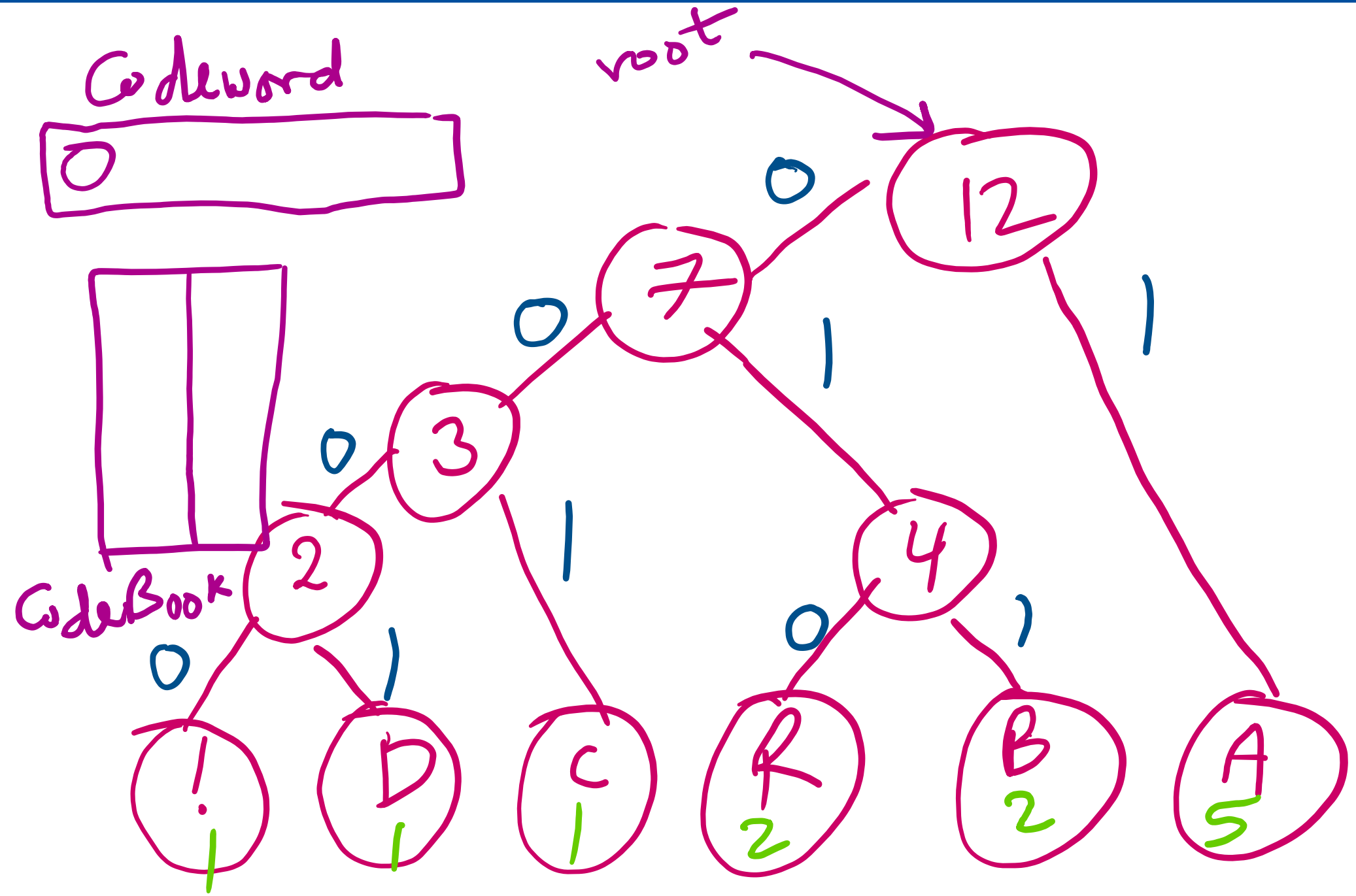
- The codebook is a table of codewords indexed by character
  - e.g., codebook[‘A’] = “1”



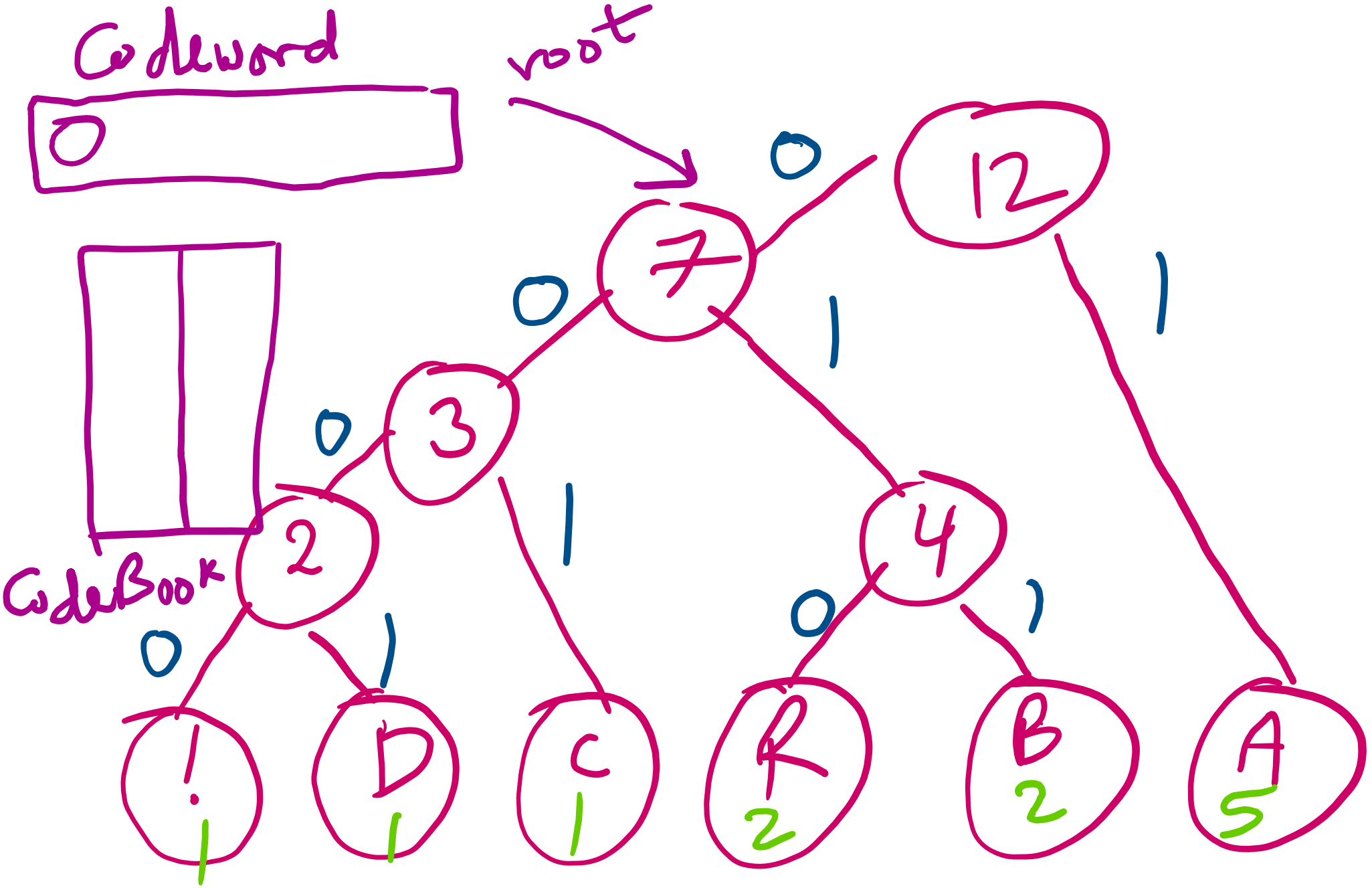
Huffman Compression: Generating the *Codebook*



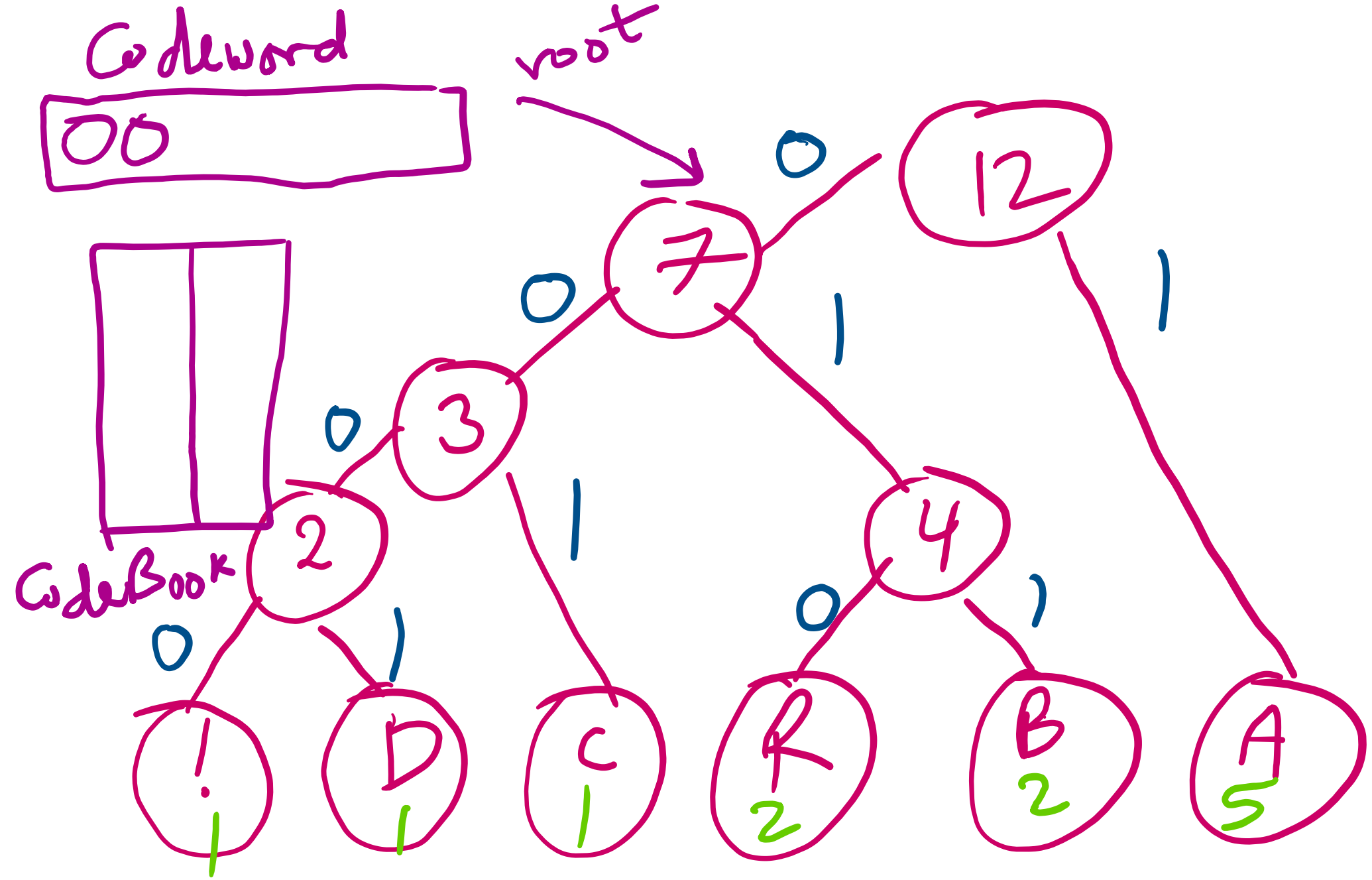
Huffman Compression: Generating the *Codebook*



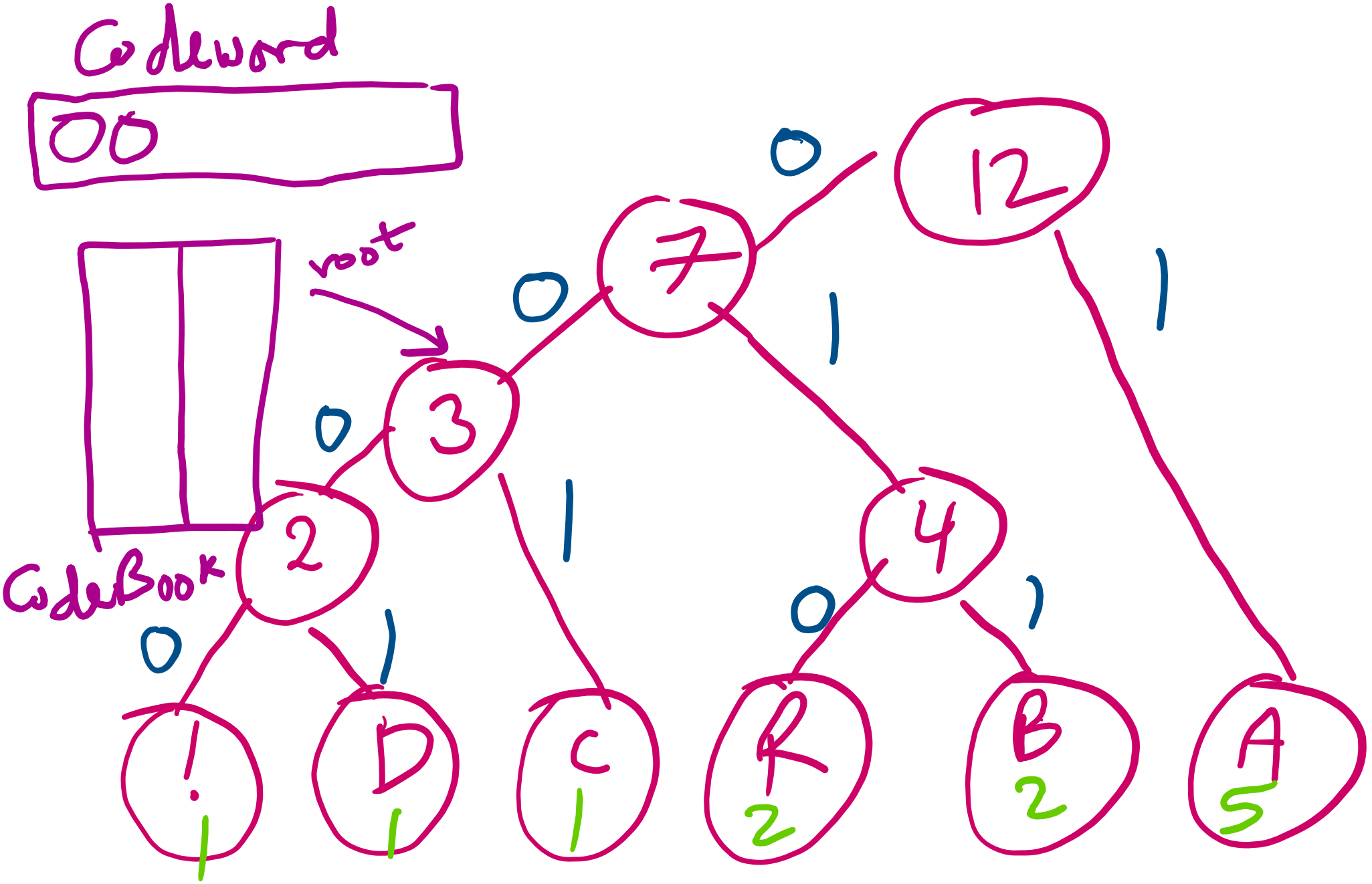
Huffman Compression: Generating the *Codebook*



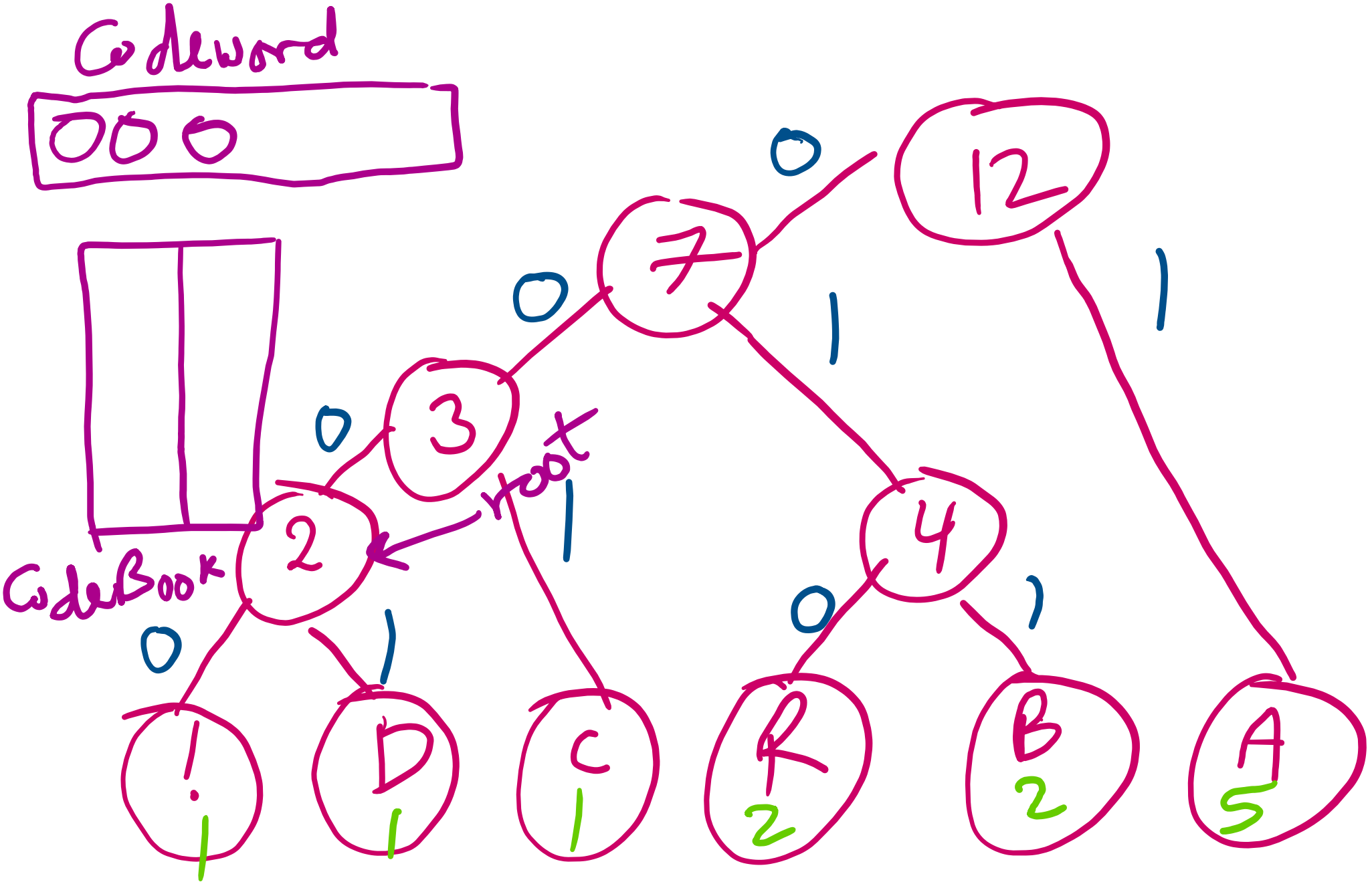
Huffman Compression: Generating the *Codebook*



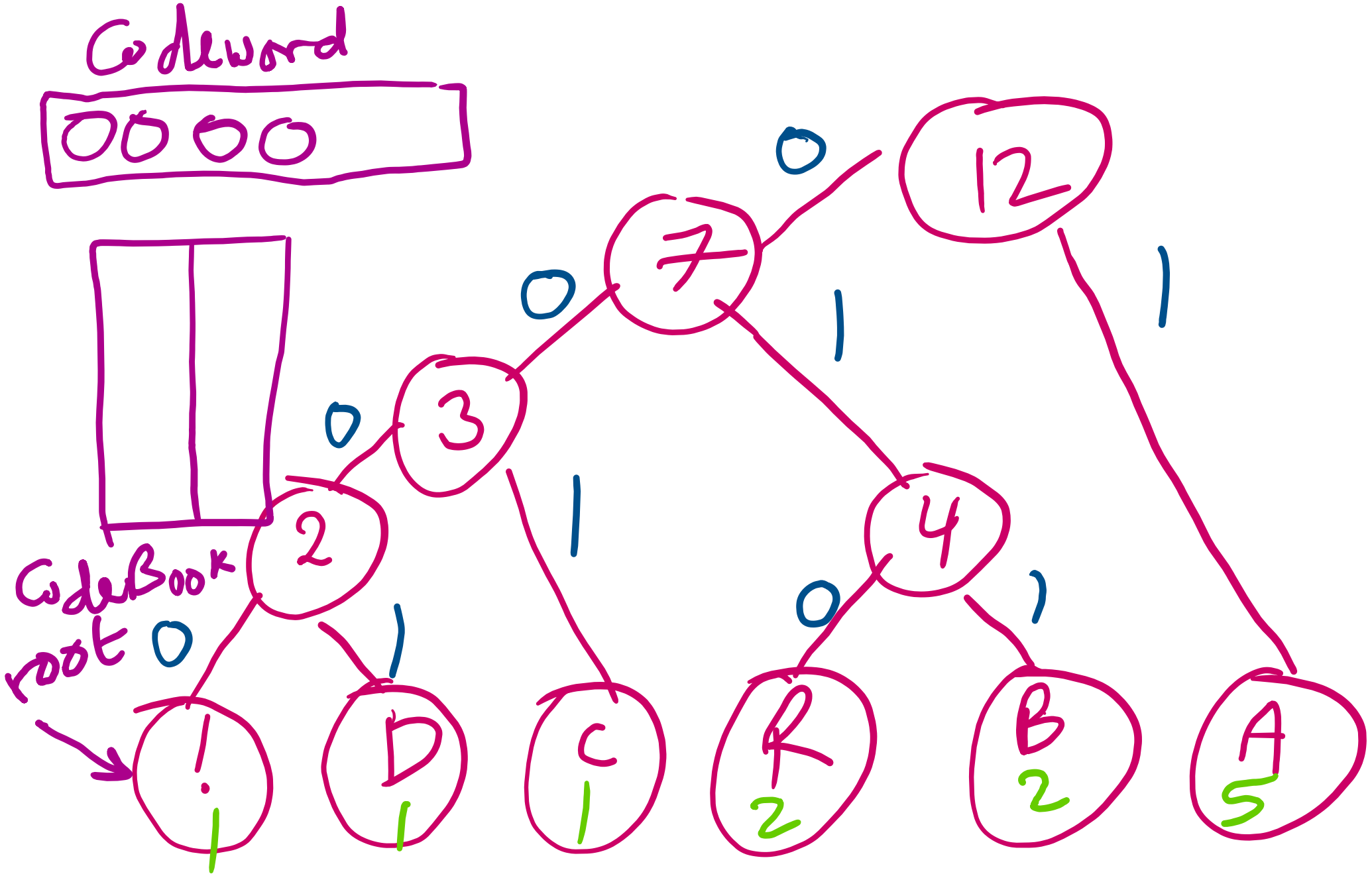
Huffman Compression: Generating the *Codebook*



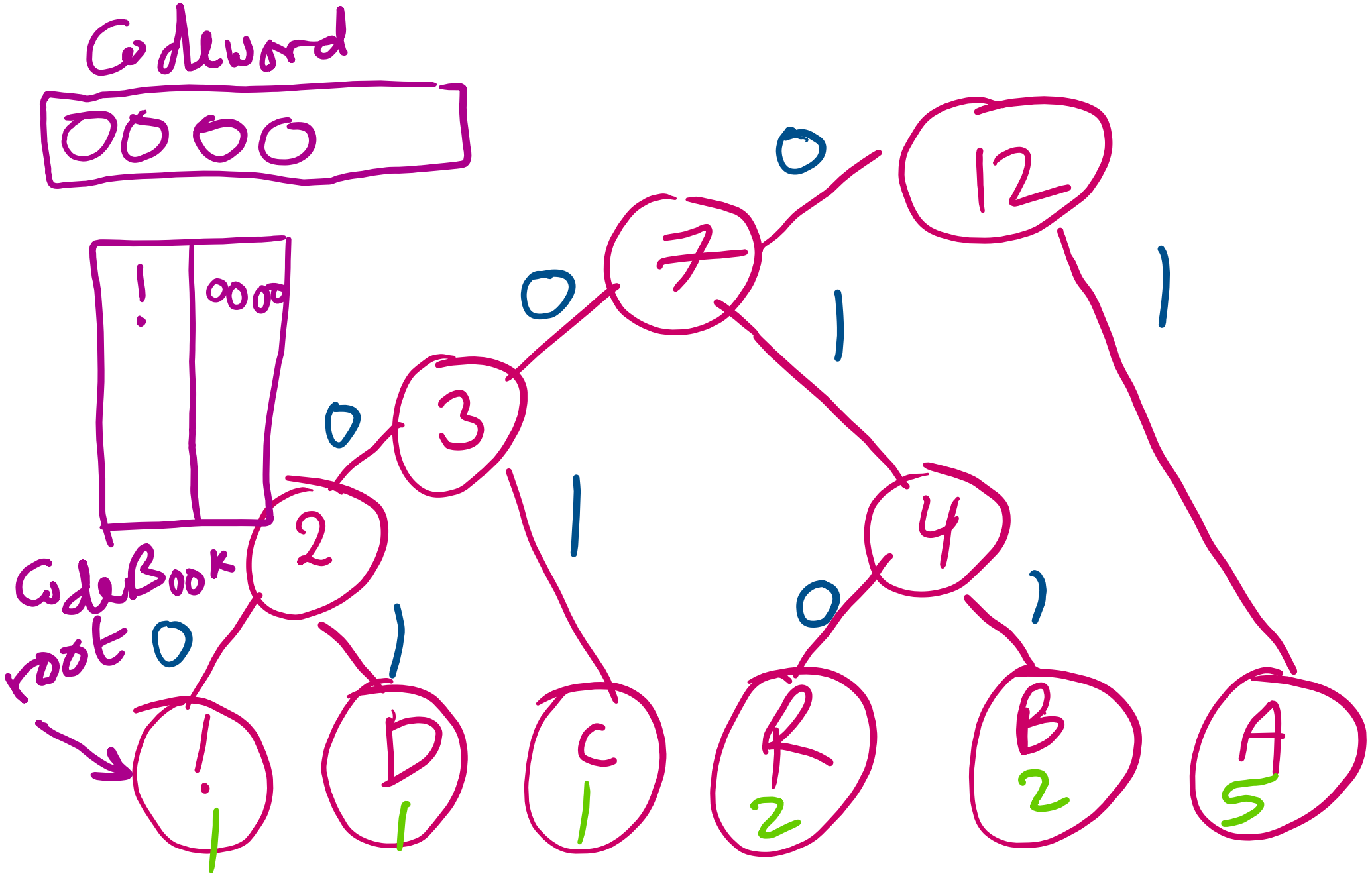
Huffman Compression: Generating the *Codebook*



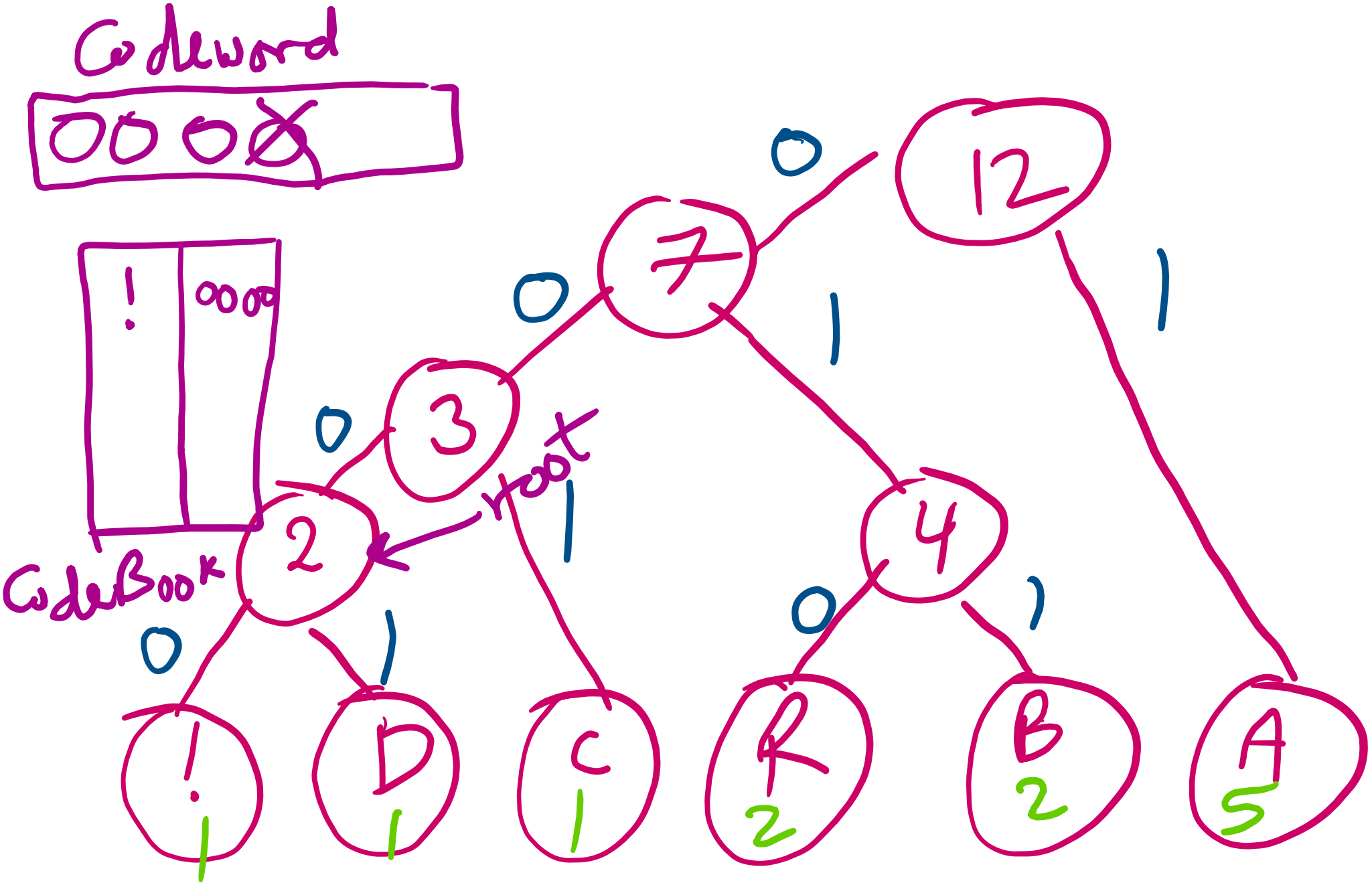
Huffman Compression: Generating the *Codebook*



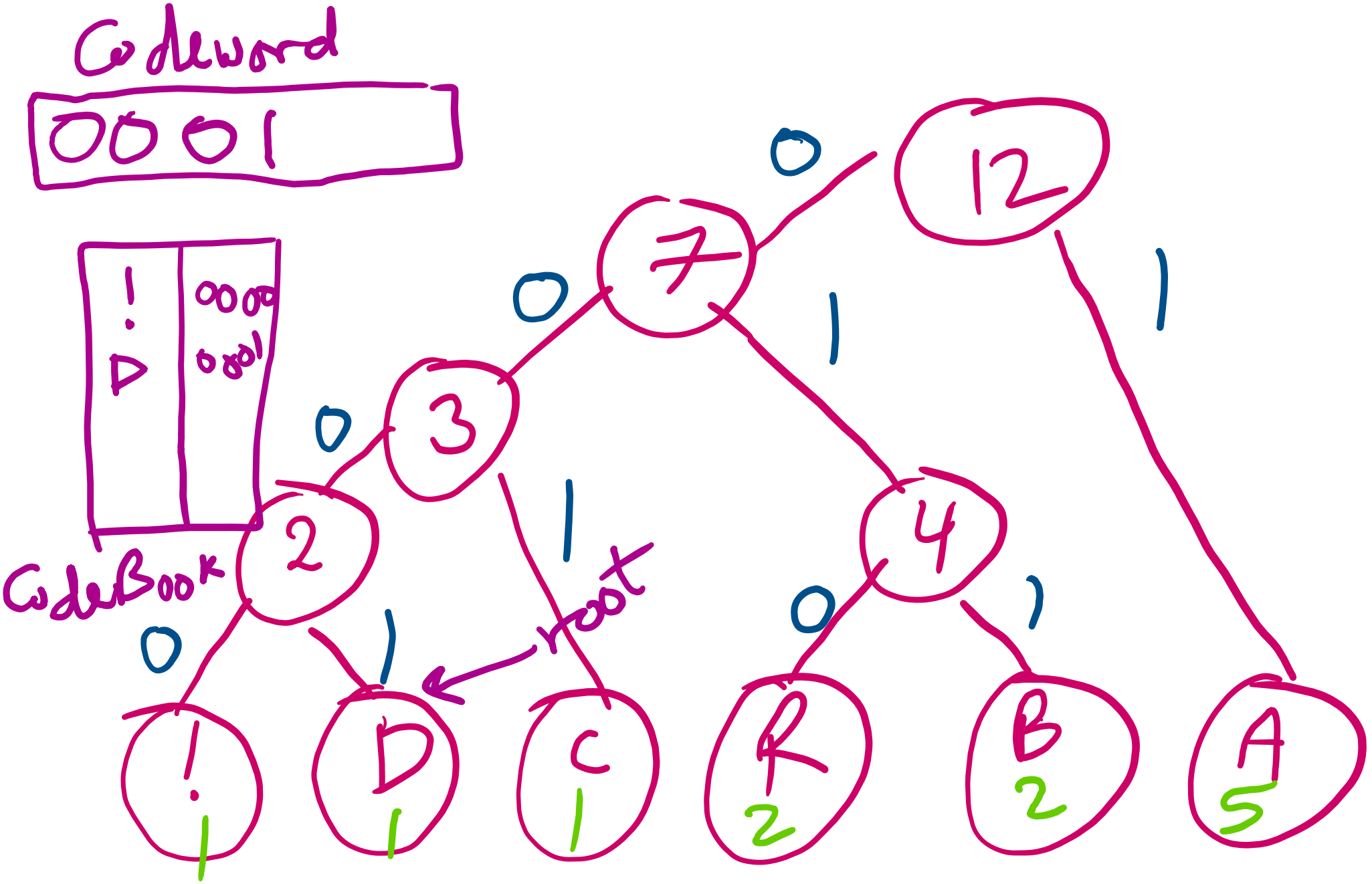
Huffman Compression: Generating the *Codebook*



Huffman Compression: Generating the *Codebook*



Huffman Compression: Generating the *Codebook*



Huffman Compression: Generating the *Codebook*

```
void generateCodeBook(Node root, StringBuilder codeword) {
    if(root.isLeaf()){
        codebook[root.data] = codeword.toString();
    }
    if(root.left != null){
        //append 0 to codeword, move left, pop last character in codeword
    }
    if(root.right != null){
        //append 1 to codeword, move right, pop last character in codeword
    }
}
```